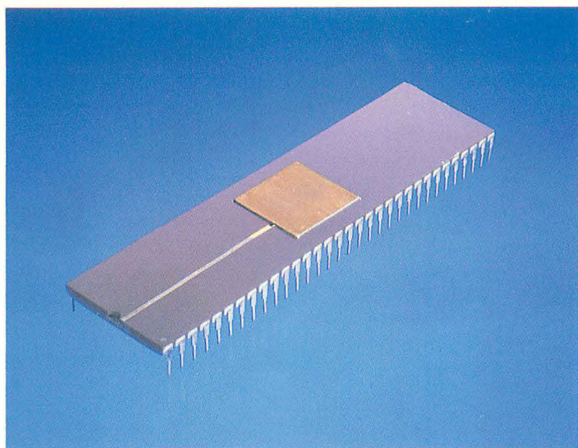
ザ 68000

●ハードウェア ●ソフトウェア ●アプリケーション

岡本 茂
佐藤 徳訓
中島 宏
大島 邦夫 共著



共立出版株式会社



上：MC 68000（日本モトローラ㈱ 提供）
下：HD 68000（㈱日立製作所 提供）

ザ 68000

●ハードウェア ●ソフトウェア ●アプリケーション

岡本 茂
佐藤 徳訓
中島 宏
大島 邦夫 共著

共立出版株式会社

主題とし，佐藤氏および中島氏によりまとめていただいた．68000 を主体とするシステムを利用しようという方には大変参考になるはずである．HD68000 を用いた各種システムについての豊富な資料を精選し，さらにその内容について十分に練り上げてある．第1章と比べれば高レベルであり，ユーザを十分に意識した立場から書かれている．

第4章は応用面についてまとめた．68000 は制御を中心とした多方面に利用されているが，ここでは制御関係は電話交換機のみとし，各種コンピュータ・システムについてまとめた．いろいろのシステムがあり，ソフトウェアもいろいろであるが，JUN のシステムは大変面白かった．これらについては関係各社の提供資料に基づき，その実体をなるべく詳しく分かりやすくまとめるように心掛けた．なお，第1章と第4章は大島氏と小生が中心となって執筆し，全体は小生が代表として取りまとめ，校正その他に当たった．

本文の脱稿後，モトローラ社からボックス・コンピュータ・システム VMC 68/2 が発表された．1983 年には VME/10 が発表されよう．これには同社と Western Electric 社が共同開発した UNIX SYSTEM V が搭載されるので VERSA dos と合わせてますます使いやすいシステムができよう．また，Apple 社からは LISA という割安なシステムが発売されている．日本ではソード社から M68，M685 が発表されている．M68 は 68000＋Z80A によるシステムで，CP/M-68 K をオペレーティング・システムにもつ．M685 は 2 つの 68000 を用いたシステムで，UNIX とコンパチブルな UNOS というオペレーティング・システムをもっている．

この企画に当たって，日本モトローラ(株)，(株)日立製作所，安立電気(株)，JUN社，YHP 社には資料の提供その他で大変お世話になった．ここに記して厚く御礼申し上げる．

また共立出版編集部の方々，特に小山透氏には企画の段階から発行に至るまで大変お世話になった．本書の完成には同氏の力が大きい．特に記して厚く感謝する．

1983 年 6 月　　著者代表　岡　本　　茂

目　　　次

1章　総　　論

2章　68000のハードウェア

3章　68000のソフトウェア

4章　応　用　例

1章　総　　論

1970年代からのLSIに関する技術進歩はめざましいものがあり，コンピュータの基本性能をLSI化した**マイクロプロセッサ**(microprocessor)が登場した．さらに技術は高度になってVLSI(超LSI)が設計製作できるようになり，1980年前後にはHMOS(high density short-channel MOS)を用いた16ビット・マイクロコンピュータが実現した．これは最新の半導体技術と優れた回路設計技術の融合によるものである．一体16ビット・マイクロコンピュータとはどんなものであろうか？

コンピュータはデータを記憶できるが，その基本となる単位は**ビット**(bit)である．これは電圧の高低などを用いて，2進数の0と1をなんらかの形で表わしたものである．ビットをいくつか集めて記憶する場所とし，それを多数集めて記憶装置――**メモリ**(memory)を作り，その一つ一つの場所に**番地**(**アドレス**；address)を付けて区別する．

たとえば4ビットであれば$2^4=16$通りのデータが表現できる．もちろんこれでは足りない場合が多いので，8ビットにして$2^8=256$通りのデータを表現するのが普通である．こうして英数字などが表現できるようになった．この単位を1**バイト**(byte)という．すなわち，この意味では1バイト＝8ビットである．1バイト＝6ビットというコンピュータもあったが，現在はたいてい1バイト＝8ビットである．このほか，さらにビット数をふやして16ビットや32ビットを1まとめにすることも考えられており，こういったものを基本単位として1個の番地を作る．

すなわち，16ビットを番地の基本単位とするものが，いわゆる16ビット・マイクロプロセッサである．

この章の主題はMotorola社が開発した16ビット・マイクロプロセッサ68000の概要であるが，8ビットのものについても触れておく．8ビットと16ビットではどこがどう違うかなどを明らかにし，68000の特徴をなるべくやさしく説明していきたい．

用語その他については，わかりやすさを第1とし，難しい用語には逐一説明を付けてあるので，この章だけでおおかたの知識が得られよう．

1.1 4ビットから8ビットへ

日本の嶋正利氏やアメリカのM. Hoff, Jr. が，1968年末に開発したIntel社の4ビット・マイクロプロセッサ4004が，日本のビジコン社で発売した電卓に組み込まれ，電卓の世界に革命が起こった．これが世界で最初のマイクロコンピュータである．当然のことながら，中に記憶されているプログラムを変えれば，コンピュータのなすべき仕事の内容を変えることができ，大変便利なので，マイクロプロセッサは広く使われるようになった．マイクロプロセッサという用語は，仕事を制御する小さな装置(LSI)を意味しており，それにクロック，RAM, ROM, 入出力用インタフェース回路，周辺機器などを加えて構成したコンピュータを，マイクロコンピュータといっている．ここにクロックは時計を意味しており，コンピュータの仕事の進行具合をコントロールする中心の時計と考えればよい．実際は数Mヘルツに及ぶ高周波を用いており，その周波数をクロック周波数という．

またRAM, ROMはそれぞれrandom access memory, read-only memoryの略で，メモリを意味している．RAMはその記憶内容を容易に変えることができるが，ROMはそうではなく，一度その内容を定めたら原則として変えられない．

入出力用インタフェースは，コンピュータと外部機器の間でデータをやりとりするための補助装置を意味しており，実際はマイクロプロセッサをいくつも使っていることが多い．

マイクロプロセッサで情報やデータをやりとりする場所を**レジスタ**という．1ビットは2進数0,1を表わすと考えてよいから，4ビットのレジスタでは2^4=16通りのデータ，すなわち1桁の16進数(0,1,2,3,4,5,6,7,8,9, A, B, C,

D, E, F) を扱える．しかし，もっといろいろなものを表わそうとするとこの4ビットでは間に合わない．そこで，ビット数を大きくすることが考えられた．たとえば，8ビットにすると $2^8 = 256$ 通りのデータが扱える．

実際，アルファベットやカナ文字はこの方法で表示されている．こうして8ビット・マイクロプロセッサが生まれ，それを利用したコンピュータが作られた．すなわち1974年にIntel社では8080，1976年にZilog社ではZ80が完成している．

Motorola社は1974年末に6800を完成させ†，その6800の機能面を強化した6801, 6802と効率面を強化した6809を作っている．6800と6809については§1.2を見られたい．

これらの8ビット・マイクロプロセッサは，いずれにしても主メモリの単位が8ビット(1バイト)であり，最大で64Kバイト（$2^{16} = 65536$）のアドレス空間をもっている．すなわち0番地から65535番地††までのアドレスをもち，各アドレスは8ビットである．

この8ビット・マイクロプロセッサにBASICをのせたパーソナルコンピュータが作られ，容易に使えるようになって，いわゆるパソコン時代がやってきた．Z80を使ったシステムとしては，日本電気(株)のPC-8001, シャープ(株)のMZ-80B2などがあり，6800を使ったものでは(株)日立製作所のベーシックマスターJr.がある．6809を使ったものでは，同社のベーシックマスターレベル3と富士通(株)のFM-8がある．

この種のコンピュータ・システムは大変多く，また，この関係の情報を載せた雑誌も多数発行されている．

もちろん8ビット・マイクロプロセッサを用いたビジネス用コンピュータ・システムも多数作られている．ビジネス用としては「使いやすく」,「プログラムの習得や作成が容易で」,「言語その他で広い適用範囲をもつ」必要があるが，これらをすべて備えることは容易ではない．

まずパーソナルコンピュータの機能を生かして，BASICをそのまま（あるいは改良したりして）利用したものが考えられた．

---

†　筆者が今までに知り得た範囲では，1974年末には，Motorola社に6800のサンプルがあったようである．

††　65535を16進数でFFFFと書くのが普通である．16進数であることを明示するため$を付けて$FFFFと書くことも多い．

一方これとは別に，在来のコンピュータ・システムで用いられてきたFORTRANやCOBOLを使いたいという要求も当然あった．8080ではシステムを操作するためのプログラム——オペレーティング・システム——としてCP/MやMP/M† などが開発されており，このシステムの上でFORTRANなどの高級言語を使うことができる．Z80は8080を拡張した形で作られていたから，CP/MやMP/MはZ80でも使える．6800や6809††に対しても，それぞれのオペレーティング・システムやFORTRANのような高級言語が開発されている．

こうしてビジネス用コンピュータ・システムが低価格で入手できるようになった．その一部はオンラインのインテリジェント端末などにも利用されている．

今日ではこれ以外に電卓，ミシン，電子レンジ，時計，電気釜，冷蔵庫その他の制御用として，マイクロプロセッサが広く使われている．ただこれらは，機械の表面で直接仕事をしているのではないから，世間の人の目に直接触れることはほとんどない．

このほか，科学技術計算でもマイクロコンピュータはよく使われている．

マイクロコンピュータの応用分野はこのように広大であり，当然いっそう高機能な16ビットに対する期待が生まれ，次の世代へと関心が移っていった．Intel社の8086，Zilog社のZ8000，Motorola社の68000，Digital Equipment Corporation (DEC) 社のLSI-11などがこうして生まれた．もちろんこれ以外にも多くの16ビット・マイクロプロセッサが開発されており，これらマイクロプロセッサを使ったマイクロコンピュータ・システムも次々と作られている．次の目標は〝完全な32ビット・マイクロプロセッサ〟の開発であるが，これも着々と進められている．

---

† CP/M＝controle program for microprocessors. マイクロプロセッサのための制御プログラムといった意味で，8080やZ80用に作られたオペレーティング・システム．大変使いやすい．

MP/M＝multiprogramming controle program for microprocessors. 多重利用ができるCP/Mともいうべきもの．どちらもDigital Research社の登録商標である．

†† 6809を用いたキヤノンのCX-1というシステムでは，COBOL，Pascalを使うこととができる．

1.2　6800 と 6809

この二つは Motorola 社の代表的な 8 ビット・マイクロプロセッサである．ここでは簡単にそのアーキテクチャ（設計思想）について述べる．

6800（図 1.1）は NMOS で作られた 40 ピンの LSI で図 1.2 に示すレジスタをもっている．

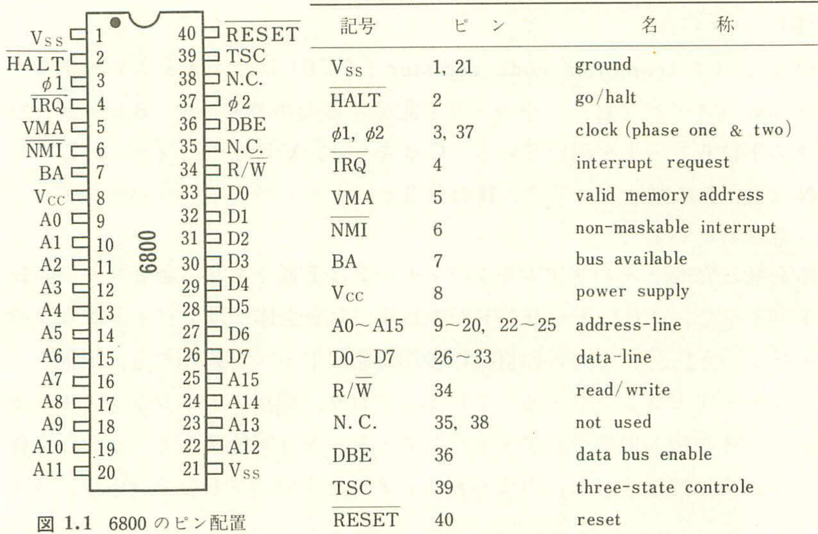


図 1.1　6800 のピン配置

記号	ピン	名　称
V_{SS}	1, 21	ground
$\overline{HALT}$	2	go/halt
$\phi 1$, $\phi 2$	3, 37	clock (phase one & two)
$\overline{IRQ}$	4	interrupt request
VMA	5	valid memory address
$\overline{NMI}$	6	non-maskable interrupt
BA	7	bus available
V_{CC}	8	power supply
A0～A15	9～20, 22～25	address-line
D0～D7	26～33	data-line
$R/\overline{W}$	34	read/write
N. C.	35, 38	not used
DBE	36	data bus enable
TSC	39	three-state controle
$\overline{RESET}$	40	reset

注．―― は低位 "low" の信号で働くことを表わす

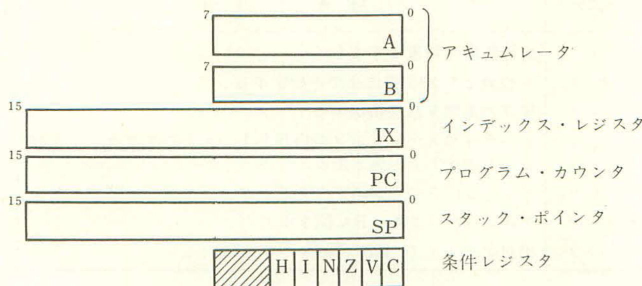


図 1.2　6800 のレジスタ構成

汎用レジスタは8ビットで2個あり，アキュムレータ(累算器)と呼ばれている．これはデータの保持や演算に用いられる．インデックス・レジスタは16ビットで，主としてメモリ・アドレスでいわゆるインデックス修飾方式をとるときに使われる(表1.1)．プログラム・カウンタは，命令の存在する番地を明らかにするために使われるレジスタで，現在実行しようとしている命令があるアドレスを指示する(この直後に次の命令のある番地のアドレスが入る)．

スタック・ポインタは，サブルーチンの戻り番地その他マイクロプロセッサの内容を退避する場所――スタック――を明示するためのレジスタで16ビットから成る．

条件レジスタ(condition code register ; CCR)は，アキュムレータで生じたあふれ(オーバフロー)やキャリを記憶するためのもので，8ビット・レジスタの下位6ビットを用いている．Cはキャリ，Vはオーバフロー，Zはゼロ，Nは負，Iは割込みマスク，Hは第3ビットからのキャリ(ハーフ・キャリ)を意味している†.

基本命令は72個，そのアドレシング・モードは7通りあり，命令コードの長さは1バイトで，これにオペランドが加わり，命令全体では1バイトから3バイトになる(図1.3)．もちろん直接に動作対象アドレスを指示でき，原則としてシングル・アドレシング・モードによっており，間接アドレシング・モードはないが，その他の大部分のアドレシング・モードは可能である．また，条件ブランチ命令は豊富で，入出力はいわゆるメモリ・マップI/Oでロード，スト

表 1.1　6800のアドレシング・モード

アドレシング・モード	対象・内容	オペランドのバイト数
イミディエート	直接数値を対象とするもの	1, 2
ダイレクト	0番地より255番地までを指定する	1
エクステンド	任意の番地を直接指定する	2
インデックス	インデックス・レジスタの内容と1バイトのオフセットを加えたものをアドレスとする	1
インプライド	インデックス・レジスタやスタック・ポインタに関するもの	0
アキュムレータ	アキュムレータA，Bに関するもの	0
リラティブ	相対アドレス(−128～127)	1

† キャリは8ビット演算による第7ビットからの桁上り，オーバフローは符号付き整数と見てのあふれを意味する．

1 バイト命令	命令コード	（インプライド・アキュムレータ）
2 バイト命令	命令コード　オペランド	（イミディエート，ダイレクト，リラティブ，インデックス）
3 バイト命令	命令コード　オペランド（2バイト）	（イミディエート，エクステンド）

図 1.3　6800 の命令フォーマット

アが入出力命令としても使われている．

6800 を中心としたマイクロコンピュータ・システムは多数開発されており，コンパイラも作られている．

6800 自身の基本周波数は，1 MHz で，これを 1.5 MHz にした 68A00, 2 MHz にした 68B00 がある．また，乗算命令などを付加した 6801 も作られている．ここではアキュムレータ A と B をこのまままとめた 16 ビット・アキュムレータ D があるのが大きな特徴で，アキュムレータ A，B を掛けてその結果を D にのせるという乗算ができる．

この 6800 を基として Motorola 社は，6802，6803 などを開発した．図 1.4 にそれらの経緯を示す．

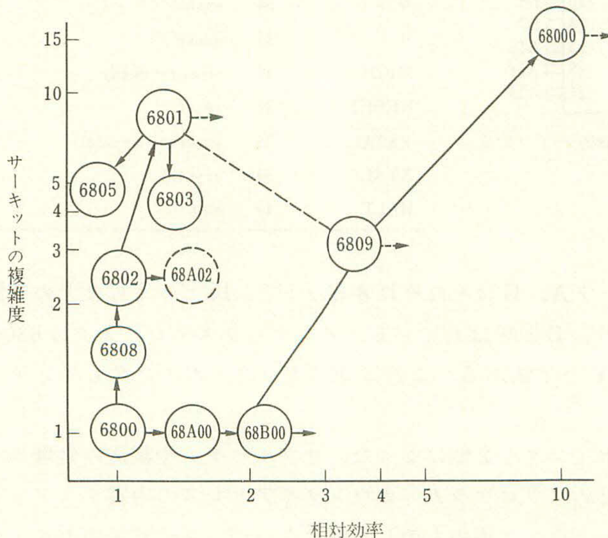


図 1.4　Motorola 社におけるマイクロプロセッサの発展

さて，6809のアーキテクチャを簡単に説明しよう．

6809は6800の機能を大幅に強化した“究極の8ビット・マイクロプロセッサ”で，16ビット演算も可能な擬似16ビット・マイクロプロセッサである．しかもHMOSを使ったLSIで，図1.5，図1.6に示すような構成になっている．

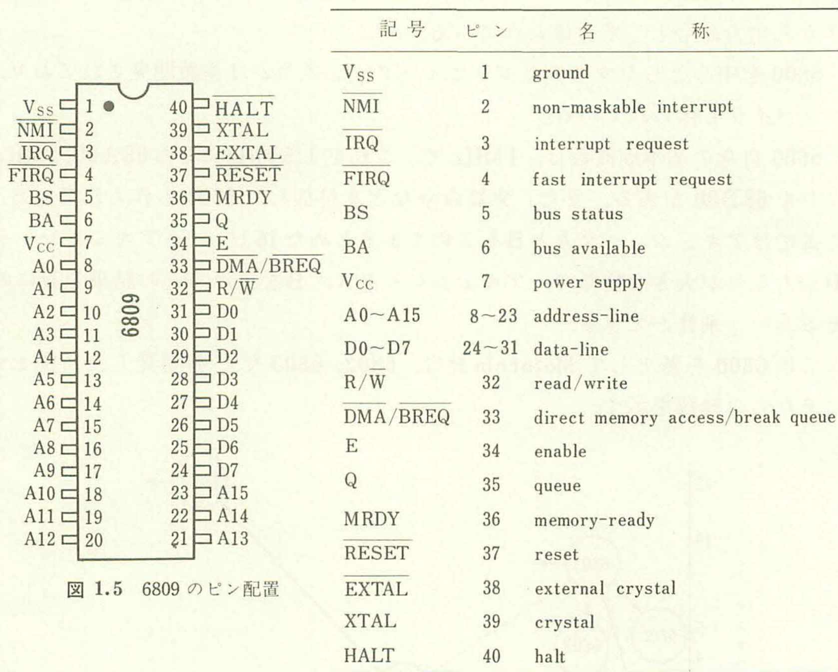


図1.5　6809のピン配置

記号	ピン	名　　称
V_{SS}	1	ground
$\overline{\text{NMI}}$	2	non-maskable interrupt
$\overline{\text{IRQ}}$	3	interrupt request
$\overline{\text{FIRQ}}$	4	fast interrupt request
BS	5	bus status
BA	6	bus available
V_{CC}	7	power supply
A0～A15	8～23	address-line
D0～D7	24～31	data-line
R/$\overline{\text{W}}$	32	read/write
$\overline{\text{DMA}}/\overline{\text{BREQ}}$	33	direct memory access/break queue
E	34	enable
Q	35	queue
MRDY	36	memory-ready
$\overline{\text{RESET}}$	37	reset
EXTAL	38	external crystal
XTAL	39	crystal
HALT	40	halt

アキュムレータA，Bはそれぞれ8ビットで，16ビットにまとめて使えるようになっており，Dと呼ばれている．インデックス・レジスタは6800より1個ふえ，IX，IYと2個ある．このほかスタック・ポインタもインデックスとして使える．

スタック・ポインタも2個になった．サブルーチンや割込み処理ルーチンへジャンプするためにプログラム・カウンタやアドレスの内容をスタックへ退避する際は，ハードウェア用のものしか使えないが，ユーザ用のポインタを利用してサブルーチンの引き数の受渡しをすることができる．したがってリエント

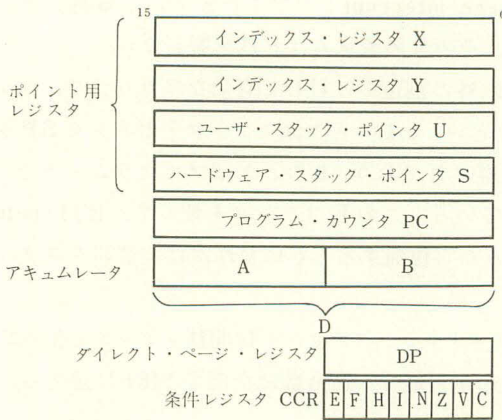


図 1.6 6809のレジスタ構成

ラントが容易にできるようになっている†.

プログラム・カウンタは6800と同じく16ビットで，アドレス空間は64Kバイトになる††.

ダイレクト・アドレシング・モード方式を強化してどこでもダイレクト・ページと同じようにするため，ダイレクト・ページ・レジスタDPが新設された．この8ビットに入っている数を上位8ビットとしてダイレクト・アドレシングが可能である．もちろんスタート時は00とセットされる．したがって256バイト以内のプログラムを使うときは大変便利である．たとえばDPに＄F0が入っていれば，ダイレクト・アドレシング・モードで＄F000～＄F0FFが指定できる．

条件レジスタCCRの下位6ビットは6800と全く同じだが，EとFの二つのフラッグが上位2ビットに新設された．

FはFIRQ (fast-interrupt request；高速割込み要求) のマスク・フラッグで，1でマスク，0でイネーブルを示している．0のときだけFIRQを受け付け，CCRをスタックにのせ，1にリセットされる．FIRQ以外に，$\overline{\text{NMI}}$ (non-maskable interrupt)，$\overline{\text{RES}}$ (reset) 信号が入力されて起動がかかるとFは

---

† たとえば，サブルーチンSを使っているときに割込みがかかり，その処理の際Sをそのまま使うというようなこと．

†† これを24ビットにしてアドレス空間を16Mバイトにし，プログラム機能の大幅な増強を行なえば本来の制御用という枠を越えたシステムが作れよう．

1になる．SWI (software interrupt；ソフトウェアによる割込み）による割込みもFを1にするが，その他の割込みはFに影響しない．

EはSWIやFIRQ以外の割込みに対して有効なフラッグで，エンタイヤ・フラッグ(entire flag)という．1のときはスタック・ポインタSPを除く全レジスタがスタックに退避されており，0のときはプログラム・カウンタPCとCCRだけが退避されている．このEフラッグを使って，RTI (return from interrupt) 命令で割込みから復帰するときにどれだけの情報をスタックから引き戻すかが決められる．

基本命令は59個，そのアドレシング・モードは間接アドレスも含めて10通りある．実に多様な命令方式が可能で，命令数は全部で1464に達する．表1.2にそのアドレシング・モードを示す．

表 1.2 6809のアドレシング・モード

モード	内　容	バイト数
インプライド	オペランドが命令だけで決まるもの．たとえばアキュムレータA，BでB+A→Bとするもの	1, 2
イミディエート	直接数値方式．数値は1または2バイト	2, 3, 4
ダイレクト	DPでページを指定し，直接アドレスする	2, 3
エクステンド	直接に16ビットのアドレスを指定する	3, 4
インデックス	IX, IY, S, U, PCをポインタ・レジスタとし，オフセットを加えて実効アドレスとする方式．11種類に分かれる	2, 3, 4, 5
リラティブ	PCの内容を中心とした相対アドレスで，ブランチだけに使われる	2, 3, 4
レジスタ	オペランド部で使用するレジスタを指定する	2
エクステンディッド・インダイレクト	間接的にアドレスを指定する	4, 5
インデックスド・インダイレクト	インデックスド・アドレシングを使用した間接アドレス方式	2, 3, 4, 5
プログラム・カウンタ・レラティブ	PCを用いた相対アドレス方式	3, 4, 5

6809の命令数は6800より少ないが，機能としては6800の命令をすべて実行でき，これを大幅に強化したものといえる．機械語ベースでは全然異なるが，アセンブラベースでは6800のプログラムを6809に直接かけられるようになっている．

なお，6809に対しては，メモリ管理ユニット6829と浮動小数用ROM6839

という周辺デバイスがあり，コンパイラやプログラムが大変作りやすくなっている．

6829は，6809のアドレス空間を64Kバイトから2Mバイトまで拡張可能にしたHMOSによるLSIで，2Kバイトから64Kバイトの範囲でタスクを分類し，1個の6829で4個のタスクをサポートできる．したがってセグメント方式が可能になった．

6839は，6809用の浮動小数点演算パッケージで，IEEE標準に合わせてあり，四則計算，剰余，平方根，整数化，絶対値，補数，正規化などの計算を実行できる．

このほかにもいろいろなLSIがあり(たとえばDMAコントローラ6844やCRTコントローラ6845など)，8ビット・マシンが容易にできるように考えられている．また6800や6809を68000の周辺LSIとして使うこともできる．

1.3　8ビットから16ビットへ

かつてマイクロプロセッサは制御に使われることが多く，4ビットや8ビットのものが大多数であった．しかし，6800から6809への進歩は，8ビットを扱えるだけでは使用分野が制限されてしまうことを示している．プログラムを作るのにも，アセンブラよりコンパイラを使うのが常識となると64Kバイトでは小さすぎる．このような状況に対応して，16ビットのマイクロプロセッサが出現し，16ビット・データを直接処理できるようになった．

一方，半導体技術の進歩はさらに加速度を増して，回路の集積度が1桁以上向上し，性能が大幅に改善された．集積回路はLSIからVLSI (very large scale integration) の時代に移り，その設計製作もCAD (computer aided design) を使わなければできないようになった†．

たとえばHMOS (high-density short-channel MOS) と従来のNMOSを比較しよう(図1.7)．NMOSでは，1セル当たり4128平方ミクロンを要するのに対し，それと同じものをHMOSで作ると1852.5平方ミクロンでよいといった具合である．この場合，速度電力積ではNMOSが約4ピコジュール，

† 1982年7月30日の読売新聞によれば，富士通(株)は，従来の2/3に縮まった64ビット・ダイナミックRAMを開発した．アクセス・タイムは100ナノ秒と20ナノ秒速くなった．

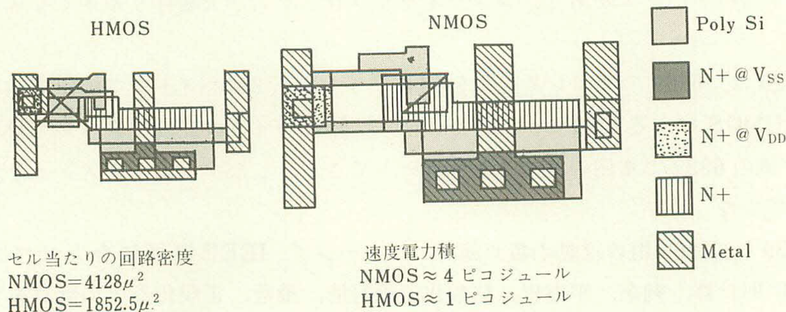


図 1.7　NMOS と HMOS の比較例 (Edward Stritter and Tom Gunter : *IEEE computer*, 12. p.43〜52, Feb (1973)).

HMOS が約1ピコジュールで, NMOS は HMOS の約4倍になっている. 技術は日進月歩であり, 今後さらに進歩していくであろう.

こういった技術の進歩を受けて16ビット・マイクロプロセッサが次々と登場してきた†. 名前だけをあげれば, Intel 社の 8086, Zilog 社の Z8001, DEC 社の LSI-11, Texas Inst. 社の 9900, Motorola 社の 68000 などがある. この

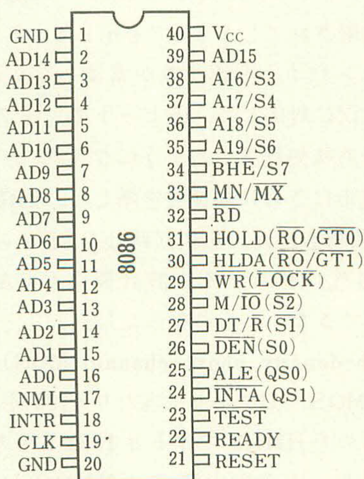


図 1.8　8086 のピン配置

† たとえば, bit 別冊"16ビット・マイクロプロセッサ", 共立出版 (1982) を見よ.

うち，8086とZ8001について多少説明をしよう．この両者はある意味で代表的な16ビット・マイクロプロセッサであり，68000ともよく比較されている．

8086は1978年前期に開発された．参考までにこのVLSIのピン配置図（図1.8）およびレジスタの概要（図1.9）を示す．

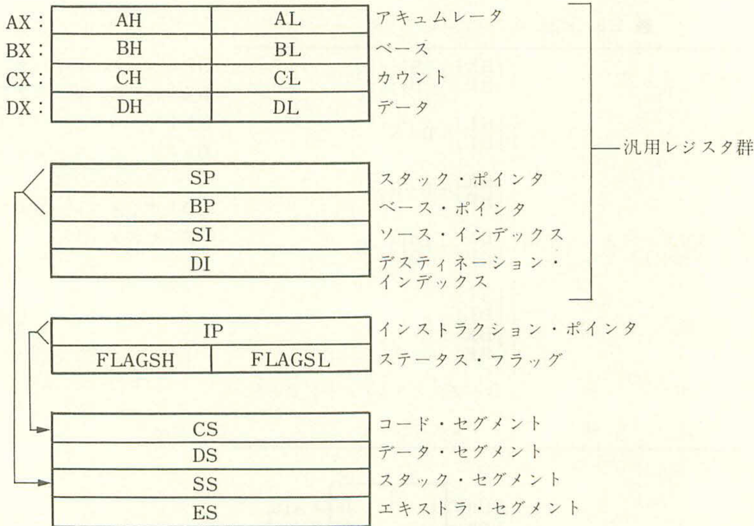


図1.9　8086のレジスタ構成

レジスタは全部16ビットで4種類に分けられ，全部で14個ある：汎用レジスタAX, BX, CX, DX，ポインタおよびインデックスSP, BP, SI, DI; プログラム・カウンタIP† とステータス・フラッグ; セグメント・レジスタCS, DS, SS, ES.

汎用レジスタはそれぞれバイトに分割して使うこともできる．命令の長さも1～6バイトにわたり，いろいろな意味でバイトを基準としている．

セグメント・レジスタにより，メモリをセグメントに分割して使うセグメント方式が可能である．この場合，プログラム・カウンタは16ビットしかないので，有効アドレスEAは「セグメントの16倍＋オフセットまたはEA」で計算され，1Mバイトのアドレス空間をもつことができる（図1.9のCS，DS，SS，ESがセグメントである）．

† Intel社では，インストラクション・ポインタといっている．

基本命令は135種あり，表1.3のアドレス方式と組み合わせて使う．ここでD＝displacementは8ビットまたは16ビットのディスプレースメントを，S＝segmentはセグメントの16倍を示している．{　}はその中の任意の一つを選んで使うことを意味する．

表 1.3　8086のアドレシング・モード

ベース・インデックス	$\left\{\begin{matrix}BX\\BP\end{matrix}\right\}+\left\{\begin{matrix}SI\\DI\end{matrix}\right\}+D+S$
インデックス	$\left\{\begin{matrix}SI\\DI\end{matrix}\right\}+D+S$
ベース	$\left\{\begin{matrix}BX\\BP\end{matrix}\right\}+D+S$
ディスプレースメントのないベース・インデックス	$\left\{\begin{matrix}BX\\BP\end{matrix}\right\}+\left\{\begin{matrix}SI\\DI\end{matrix}\right\}+S$
間接	$\left\{\begin{matrix}SI\\DI\\BX\\BP\end{matrix}\right\}+S$
相対	D＋プログラム・カウンタ＋S
直接	アドレス＋S

BP：ベース・ポインタ
BX：ベース・レジスタ
SI：ソース・インデックス
DI：デスティネーション・インデックス
D：ディスプレースメント
S：セグメント

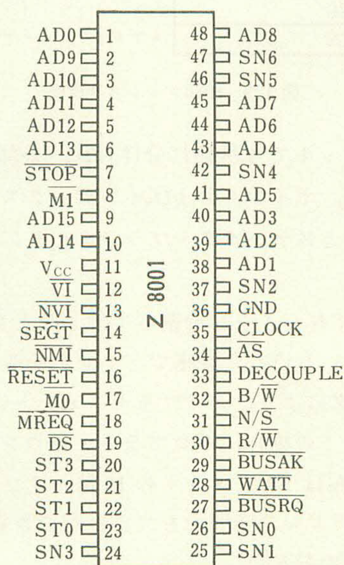


図 1.10　Z 8001のピン配置

ソフトウェアもよく完備しており, Digital Research 社の CP/M-86†や Bell 研究所の UNIX†のもとに走るコンピュータ・システムがある．もちろんアセンブラや FORTRAN, BASIC, FORTH なども使えるようになっており, 日本でもいろいろなシステムが発売されている．

次に Z8001 を簡単に紹介しよう．これは1979年始めに開発されていたようである．そのピン配置図を示す(図1.10)．

レジスタは全部16ビットで, 汎用が14本, スタック・ポインタやプログラム・カウンタなどが7本ある．このうち, 8本の汎用レジスタ R0-R7 はバイトに分割して使える．命令は1～5ワードの長さをもち, 必ずワード位置アドレスすなわち偶数バイト・アドレスから始まる．この意味で Z8001 は16ビット・マイクロプロセッサであり, 8086のようなバイト意識の強いものとは異なる(図1.11)．

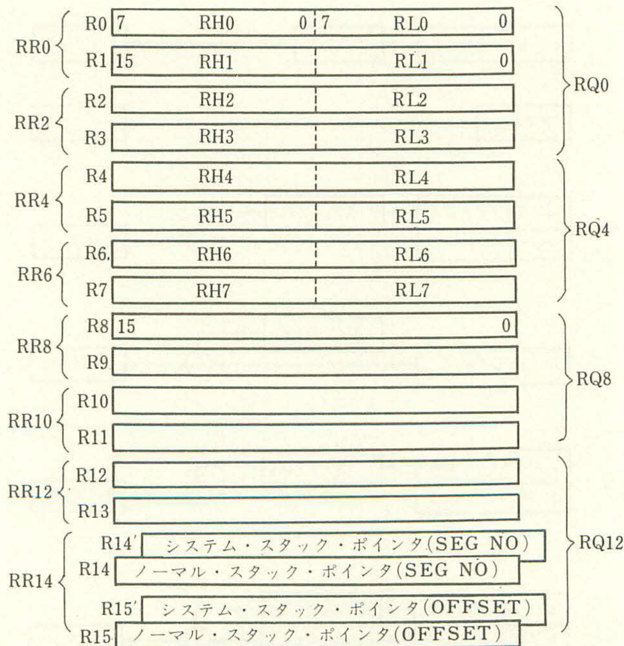


図 1.11　Z8001 のレジスタ構成

† CP/M は Digital Research 社の登録商標, UNIX は Bell 研究所 (ATT) の登録商標である．

セグメント方式も可能で†，7 ビットのセグメントの値と 16 ビットのオフセットを使ってアドレスを決めるようになっており，8 M バイトのアドレス空間をもつことができる．

基本命令は約 170 種あり，いろいろなアドレス方式と組み合わせて使う．これは汎用レジスタをベース・アドレス，インデックス，ディスプレースメント，絶対アドレス，カウンタに用いており，8086 とは異なる．8086 と比較できる

表 1.4　8086 のアドレシング・モード

モード	オペランドの形式 命令内	 レジスタ内	 メモリ内	オペランドの値
レジスタ	レジスタ・アドレス →	オペランド		レジスタの内容
即　値	オペランド			命令の中にある数
レジスタ間　接	レジスタ・アドレス →	アドレス →	オペランド	レジスタにあるアドレスが指すメモリ
直　接アドレシング	アドレス →		オペランド	命令の中にあるアドレスが指すメモリ
インデックス	レジスタ・アドレス → ベース・アドレス →	ディスプレースメント → ＋	オペランド	命令の中の値とレジスタの内容との和で決まるアドレスが指すメモリ
相　対アドレシング	ディスプレースメント →	PC の値 → ＋	オペランド	プログラム・カウンタと命令の中のディスプレースメントの和で決まるアドレス
ベース・アドレシング	レジスタ・アドレス → ディスプレースメント →	ベース・アドレス → ＋	オペランド	レジスタの内容と命令の中の値との和で決まるアドレスが指すメモリ
ベース・インデックス	レジスタ・アドレス → レジスタ・アドレス →	ベース・アドレス ディスプレースメント → ＋	オペランド	二つのレジスタと命令の中のディスプレースメントの和で決まるアドレスが指すメモリ

† セグメント方式のない Z8000 を Z8002 という．これは 64 K バイトのアドレス空間をもっている．

ように表1.4と表1.5に示す.

UNIX やその拡張をオペレーティング・システムにもつものがすでにいくつか発売されている. COBOL や FORTRAN, UCSD Pascal[†] なども使えるようになった.

命令の内容その他については述べなかったが, 詳細を知りたい人はマニュアルなどを参考にされたい.

表 1.5 Z8001 のアドレシング・モード

モード	オペランドの形式: 命令内	オペランドの形式: レジスタ内	オペランドの形式: メモリ内	オペランドの値
レジスタ	レジスタ・アドレス	オペランド		レジスタの内容
即値	オペランド			命令の中にある数
レジスタ間接	レジスタ・アドレス		オペランド	レジスタで指定されたアドレスが指すメモリ
直接アドレシング	アドレス		オペランド	命令内で指定されたアドレスのメモリ
インデックス	レジスタ・アドレス ベース・アドレス	ディスプレースメント	⊕ → オペランド	レジスタの内容をディスプレースメントとし, それをベース・アドレスに加えたアドレス
相対アドレシング	ディスプレースメント	PCの値	⊕ → オペランド	命令内にあるディスプレースメントをプログラム・カウンタの内容に加えたアドレス
ベース・アドレシング	レジスタ・アドレス ディスプレースメント	ベース・アドレス	⊕ → オペランド	命令内にあるディスプレースメントを, レジスタ内にあるベース・アドレスに加えたアドレス
ベース・インデックス	レジスタ・アドレス レジスタ・アドレス	ベース・アドレス ディスプレースメント	⊕ → オペランド	レジスタ内にあるディスプレースメントを, レジスタ内にあるベース・アドレスに加えたアドレス

† カリフォルニア大学理事会の登録商標である.

1.4　68000の概説

6800ファミリを開発したMotorola社は，その16ビット化をめざして，次のマイクロプロセッサの開発に入った．その一つの表われが“究極の8ビット・マイクロプロセッサ”6809で，いわゆる擬似16ビットを目標としたものである．

もう一つは完全な16ビット・マイクロプロセッサで，この開発に当たりDEC社のPDP-11をはじめとして，いろいろなミニコンピュータを調べたようである．このような研究を基として32ビット・マイクロプロセッサ68020に到達し，その簡約化として68010と68000が開発された．68000は32ビットの内部アーキテクチャをもつ“最初のマイクロプロセッサ”である．

Intel社には，8086を基とし，そのデータ・バスを8ビットにした8088がある．16ビット・マイクロプロセッサの8ビット化には8ビット・マイクロプロセッサとは異なる意味があると考えたMotorola社は68008を開発した．

もちろんこれらの周辺デバイスも開発されており，今日では68000ファミリによる優秀なマイクロコンピュータ・システムが作られている．

そのアーキテクチャは高度に系統化され，レジスタの個数も6800より大幅にふえ，バイト，ワード，32ビット・ワードを直接扱うことが可能になった．モジュラ・プログラミングも容易でそのための命令もある．したがって構造化アセンブラやPascalのようなブロック構造をもった高級言語を容易に使うことができる．

68000は独特のマルチレベル・マイクロプログラム構造をもっており，将来の増強に備えてOPコードに特別な空きがある．利用者は現在の命令セットにない命令を，トラップ命令とエミュレータ・トラップを使って作成し実行できる．

命令の処理速度は，マイクロプロセッサの大変重要な因子で，このため命令コードは徹底的に検討されたようである．実際にどれだけの頻度で命令が実行され，プログラム・リストにどんな命令がよく使われるかを調べた結果，命令の処理能力向上には「高度に系統化されたアーキテクチャ」が役立つことがわかり，命令セットやアドレシング・モードに大幅な改善が加えられ，56種の基本命令と14種類のアドレシング・モードが決まった．

一方，HMOSを用いてVLSIの大きさや処理速度を改良した．基本クロックも4MHzから始まって6MHz，8MHz，10MHz，12.5MHzと向上している．たとえば，レジスタをクリアする命令のような最も短い命令実行時間は，4サイクルである．これは4MHzでは1μ秒，12.5MHzなら320ナノ秒に相当する．

このほか，メモリ管理にも細心の注意が払われている．8086やZ8001と異なり，68000は16Mバイトのアドレス空間をもつことができ，さらにそのどこでも直接アクセスするように命令を組むことができる．

またソフトウェアも良くなり，オペレーティング・システムもいろいろのものが使えるようになった．もちろん，BASIC，FORTRAN，PL/Iなど多くのコンパイラがあり，大変使いやすいコンピュータ・システムを構成できる．図1.12，図1.13と表1.6にそのアーキテクチャの概要をまとめた．

レジスタは一体何個あったらよいのだろうか？ずっと前のミニコンでは，大抵のレジスタは，インデックスを除いて，一個しかなかった．ある場合には不便であったので，それを何とかしようとしてふえたようにみえる．

レジスタを大別すると次のようになろう：アドレス関係，データ関係，インデックス，プログラム・カウンタ，スタック・ポインタ，その他．そこで命令部を4ビットとしてしまうと，（16ビット・マイコンでは）12ビットをオペランドに振り向けられるので，シングル・アドレシング・モードでこの12ビットをオペランドに向けるか，2-アドレシング・モードで6ビットずつに分けるか，3-アドレシング・モードで4ビットずつに分けるかが問題になる．もし命令部がもっと大きければ，当然オペランド部を縮めなければならないか，またはもう1バイトないし2バイトを使って表現するようにしなければならない．Motorola社では2-アドレシング・モードを選び，6ビットによってレジスタを指定するようにした．詳しくは25ページの図1.18をみられたい．

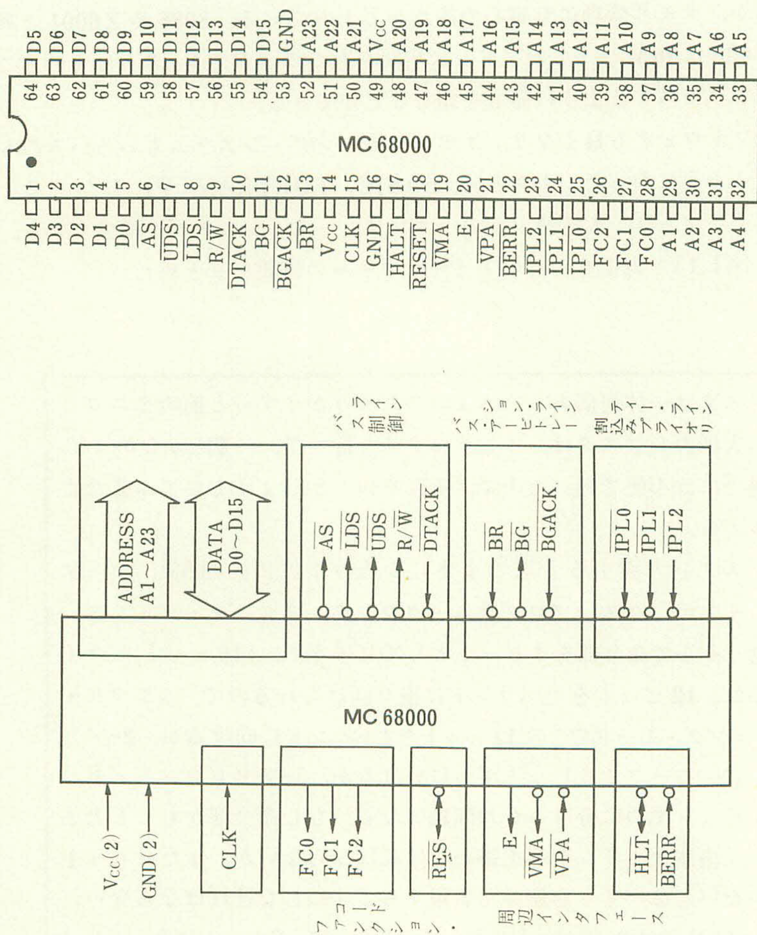


図 1.12　68000 の入出力信号・ピン配置・各部の名称

記 号	ピ ン	名	称
D0~D15	1~5, 54~64	data-line	asynchronous bus control
$\overline{\mathrm{AS}}$	6	address-strobe	
$\overline{\mathrm{UDS}}$	7	uper-data strobe	
$\overline{\mathrm{LDS}}$	8	lowe-data strobe	
R/$\overline{\mathrm{W}}$	9	read/write	
$\overline{\mathrm{DTACK}}$	10	data transfer acknowledge	
$\overline{\mathrm{BG}}$	11	bus grant	bus arbitration control
$\overline{\mathrm{BGACK}}$	12	bus grant acknowledge	
$\overline{\mathrm{BR}}$	13	bus request	
V_{CC}	14, 49	power supply	
CLK	15	clock	
GND	16, 53	ground	
$\overline{\mathrm{HALT}}$	17	halt	system control
$\overline{\mathrm{RESET}}$	18	reset	
$\overline{\mathrm{BERR}}$	22	bus error	
$\overline{\mathrm{VMA}}$	19	valid memory address	M6800 peripherel control
E	20	enable	
$\overline{\mathrm{VPA}}$	21	vilid peripheral address	
$\overline{\mathrm{IPL}}$	23~25	interrupt control	
FC	26~28	processor status	
A1~A23	29~48, 50~52	address-line	

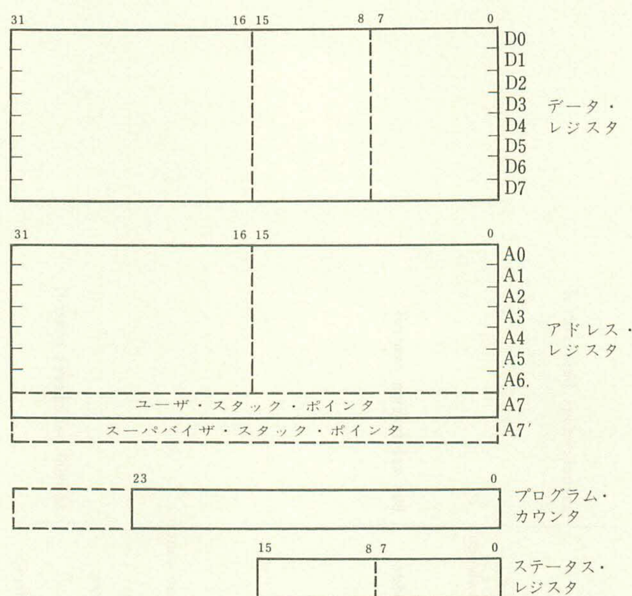


図 1.13　68000 のレジスタ構成

表 1.6　68000 のレジスタの概要

名　称	個数	記　号	ビット数	備　　考
データ・レジスタ	8	D0, …, D7	32	8, 16, 32 ビットとして使える
アドレス・レジスタ	7	A0, …, A6	32	16, 32 ビットとして使える
スタック・ポインタ	2	USP, SSP	32	USP：ユーザ・スタック・ポインタ SSP：スーパバイザ・スタック・ポインタ
プログラム・カウンタ	1	PC	32	下位 24 ビットのみ使う $2^{24}=16\times1024\times1024$
ステータス・レジスタ	1	SR	16	システム・バイトとユーザ・バイトに分かれる

データ・レジスタかアドレス・レジスタのいずれかをRnで表わすことがある．すなわち，Rn＝An またはDn とするのである．たとえばRm と Rnの内容を交換する命令 EXG では，この記号を使って説明するのが普通である（アセ

ンブラでEXG Rm, Rnと書く).

ステータス・レジスタ(図1.14)は下位8ビットをユーザ・バイトとし，条件レジスタとしている．上位8ビットはシステム・バイトである．

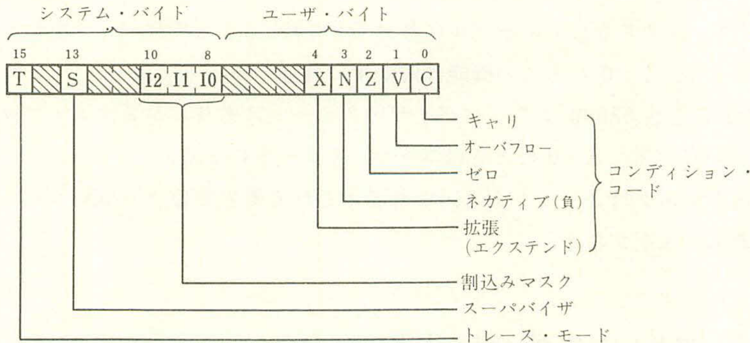


図 1.14 ステータス・レジスタの構成

条件コードとして拡張X，ネガティブN，ゼロZ，オーバフローV，キャリCがある．上位3ビットは0として読まれ，現在は使われていない．将来の機能拡張に備えているものと解される．計算を行なった結果生じた状態を記録するもので，次のような意味をもっている．

ビット0．キャリC：加算の結果生じた桁上げ，減算の結果生じた借り(ボローという)およびシフトで生じたビットの状態を記録する．比較によっても変化する．こういう状態が生ずると1になり，生じなければ0となる．

ビット1．オーバフローV：符号をこめた演算で，表現できる結果を超えるときだけ1になる(最上位とその隣りのビットで生じたものの排他的論理和)．シフトで生じることもある．

ビット2．ゼロZ：結果が0のとき1になる．

ビット3．ネガティブN：符号をこめた演算で，結果が負のときだけ1になる．シフトで生じることもある．

ビット4．拡張X：多倍長演算におけるキャリ．加減算，補数，シフトで有効である．

システム・バイトはトレース・モードT，スーパバイザS，割込みマスクI0，I1, I2から成り，他のビットは0として扱われ，使われていない(拡張に備え

ているものと解される).

T=1 だと 68000 はプログラムをシングル・ステップ実行する．すなわち一命令を実行するたびにスーパバイザ・ステートに入り，ユーザの作ったトレース・サービス・ルーチンにとぶ（サービス・ルーチンのエントリ・アドレスはトラップ・ベクタとしてテーブルに与えておく)．こうしてデバッグを行なうことができる．T=0 だとこの機能は働かず，普通のモードで動く．

S=1 のとき 68000 はスーパバイザ・ステートにあり，システム・プログラムが働いている．S=0 のときはユーザ・ステートになる．

割込みマスクは 3 ビットあり，これで示される優先順位より低いレベルの割込み要求は無視される．

1.5 アドレシング・モードおよび命令

68000 のアドレス空間は，バイトによりアドレス付けされ，たとえば 0 番地と 1 番地のバイトをまとめて 0 番地のワードと考えている．一般に 2n 番地と 2n+1 番地をまとめて 2n 番地のワードと考えるので，ワードは 0 番地，2 番地，4 番地，…と番地付けられる．奇数番地のワードはない．

F E D C B A 9 8 7 6 5 4 3 2 1 0

ワード 000000	
バイト 000000	バイト 000001
ワード 000002	
バイト 000002	バイト 000003
⋮	⋮
ワード FFFFFC	
バイト FFFFFC	バイト FFFFFD
ワード FFFFFE	
バイト FFFFFE	バイト FFFFFF

図 1.15　メモリにおけるワードの構成

命令の対象となるデータは，1，8，16，32 ビットの整数，アドレス・データおよび BCD データである．BCD データ（図 1.16）は 2 進化 10 進数によるデータであって，4 ビットで 1 桁の 10 進数を表わし，原則として 32 ビット（8 桁）である．

15	11	7	3	0
BCD0	BCD1	BCD2	BCD3	
BCD4	BCD5	BCD6	BCD7	

図 1.16 BCD データの構成

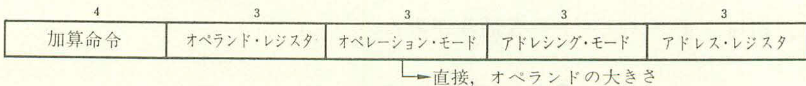
一般に，命令は「実行すべき機能」を指定する**命令部**とその対象になる「データまたはデータのある場所に関する情報」を書いた**オペランド**から成る．

命令の長さは 1 ワードから 5 ワードで，その最初のワードで命令部が指定され，それを除いた部分がオペランドを表わす（図 1.17）．具体的な形はここでは述べないが，その例をいくつかあげる（図 1.18）（第 2 章参照）．

オペレーション・ワード	（1 ワード）
イミディエート・オペランド	（0，1 または 2 ワード）
ソース実効オペランド	（0，1 または 2 ワード）
目的実効オペランド	（0，1 または 2 ワード）

図 1.17 命令のフォーマット

レジスタ同士の加算（数はビット数を表わす）



インデックス・レジスタを用いた加算

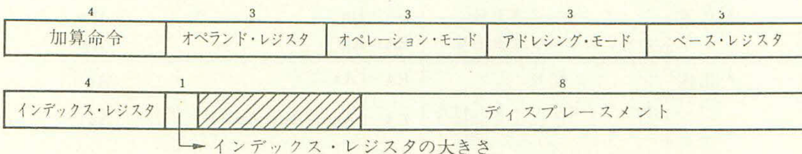
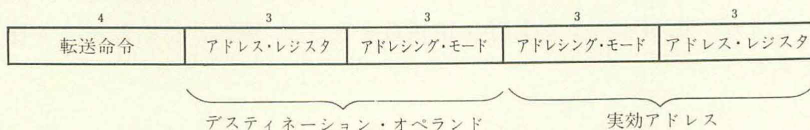


図 1.18 命令のフォーマットの例



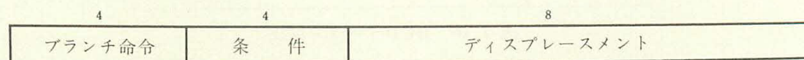


図 1.18（続き）　命令のフォーマットの例

さて，オペランドについてまとめよう．ほとんどの命令では，オペランドが実効アドレスを意味しているが，間接やインデックスを使ったときはそうではない．このアドレス方式は全部で14種ある．一般に汎用レジスタAmやDnには0から始まる番号が付けられており，3ビットで表現できる．図1.18に示した形を次のようにまとめる．

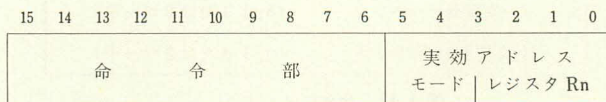


図 1.19　オペレーション・ワードの例

アドレシング・モードにはレジスタ直接，レジスタ間接，絶対，イミディエート，相対，インプライドの5通りがある．

表 1.7　68000のアドレシング・モード

基本型	モード	内容	アセンブラ
レジスタ直接	データ・レジスタ直接	EA＝Dn	Dn
	アドレス・レジスタ直接	EA＝An	An
レジスタ間接	レジスタ間接	EA＝(An)	(An)
	ポストインクリメント付きアドレス・レジスタ間接	EA＝(An)，An←An＋N	(An)＋
	プリデクリメント付きアドレス・レジスタ間接	An←An－N，EA＝(An)	－(An)

レジスタ間接	ディスプレースメント付きアドレス・レジスタ間接 インデックス・レジスタ付きアドレス・レジスタ間接	EA=(An)+d16 EA=(An)+(Ri)+d8	d(An) d(An, Ri)
絶　対	短いもの 長いもの	EA=(命令の次の1語) EA=(命令の次の2語)	×××× ××××××××
イミディエート	イミディエート クイック・イミディエート	データ=命令の次の1語 インヒアラント（固有データ）	#×××× #××
相　対	オフセット付き相対 インデックス付き相対	EA=(PC)+d16 EA=(PC)+(Ri)+d8	d16 d(Ri)
インプライド	インプライド・レジスタ	EA=SR, USP, SP, PC	

EA=実効アドレス
d8, d16=8または16ビットのオフセット
N=1(バイト), 2(ワード), 4(ロングワード)
( )=内容を表わす
←=置換え
Ri=インデックス・レジスタとして使われたAiまたはDi
SR=ステータス・レジスタ
PC=プログラム・カウンタ
USP=ユーザ・スタック・ポインタ
SP=現在使われているスタック・ポインタ

以下においてこれら68000のアドレシング・モードを説明しよう．16進数を表わすために$を用いるので注意してほしい．これはMotorola社の記法に従ったためである．たとえば$123Fは16進数の123F，すなわち$1\times16^3+2\times16^2+3\times16+15=4671$を意味する．また{ }は( )内の長さだけとることを示す．

1.5.1 レジスタ直接

命令形式は次の通りで，その長さは1～3ワードである．任意の汎用レジスタRnを実効アドレスとするもので，これによって68000を16個のレジスタから成る電卓として扱うこともできる．モードは2進数で表わされている．

命　令	モード	n
{目的オペランド(長さ 0, 1, 2)}		

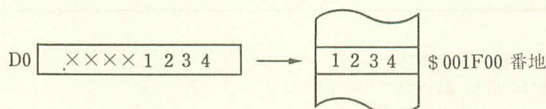
(モード 000 : Dn
　　　　001 : An)

【例1】 MOVE.L D0, A4　　これはレジスタD0の内容をA4に転送せ

よという(アセンブラ形式の)命令で，機械語では $2840 という形に書ける．.L は 32 ビットで動作せよという意味である．

0 0 1 0	1 0 0 0 0 1	0 0 0 0 0 0
L	目的アドレス	実効アドレス

【例 2】 MOVE.W　D0，$1F00　　これはレジスタ D0 の内容をワード単位で $1F00 番地に転送せよという命令で，$31C0 1F00 という機械語になる．.W はワードすなわち 16 ビットで動作せよという意味である(バイトの場合は .B と書く．省略するとワードの意味に解され，.W があったものとする)．



【例 3】 MOVE　$201000，A4　　これは $201000 番地の内容をワード単位で A4 に転送せよという命令で，アドレス・レジスタ A4 では頭のビット 15 を符号とみなし，これを付けた形で下 16 ビットに受け入れる(これを自然な符号拡張という)．もし $201000 番地の内容が $1234 なら，命令実行後の A4 の内容は $00001234 になる．もし $201000 番地の内容が $8000 なら A4 は $FFFF8000 になる．この命令は $387900201000 と書かれ，3 ワードを占める．

1.5.2 アドレス・レジスタ間接

アドレス・レジスタ An の内容をアドレスとみなし，これを実効アドレスとするもの．命令形式は次の通りで，その長さは 1～3 ワードである．

命　　令	010	n
{目的オペランド (長さ 0, 1, 2)}		

【例 1】 MOVE　(A0)，D0　　もしアドレス・レジスタ A0 の内容が，$001000 であれば，$001000 番地の内容が D0 に転送される．すなわち命令としては D0←((A0))．

【例 2】 MOVE　(A4)，$1F00　　これは $001F00 番地に A4 で指定した番地の内容を転送する命令である．もし A4 の内容が $1000 なら，$001000 番地の内容を $001F00 番地に転送する．

1.5.3 ポスト・インクリメント付きアドレス・レジスタ間接

アドレス・レジスタAnの内容をアドレスとみなし，これを実効アドレスとするのは§1.5.2と同様だが，データ操作が実行された後，Anの内容がN（＝1, 2または4）だけふえる．すなわちAn←An+N．Nは「データの大きさ」を示すバイト数である(ポストpostは“～の後で”という意味の接頭語).命令形式は次の通りで，その長さは1～3ワードである．

命　　令	011	n
{目的オペランド（長さ 0, 1, 2)}		

【例1】 MOVE (A4)+, $2000　　もしA4の内容が$00001000なら，$1000番地の内容を$2000番地に転送し，A4は$00001002になる．

1.5.4 プリデクリメント付きアドレス・レジスタ間接

これも§1.5.2と同じような動作をするが，最初にNだけAnを減らし，その上で「レジスタ間接」を行なう．すなわち指定されたアドレス・レジスタAnの内容を「データの大きさ」だけ減らし(An←An−N)，その上でデータ操作を行なう（プリpreは“～の前に”という意味の接頭語)．命令形式は次の通りで，その長さは1～3ワードである．

命　　令	100	n
{目的オペランド（長さ 0, 1, 2)}		

【例1】 MOVE −(A3), $4000　　まずA3の内容を2だけ減らし，新しいA3の内容を番地とみなしてその番地の内容を$4000番地に転送する命令である．たとえば，A3の内容が$00000100なら新しいA3は$000000FEになり，$FE番地の内容が$4000番地に転送される．

1.5.5 ディスプレースメント付きアドレス・レジスタ間接

命令形式は次の通りで，長さは2～4ワードである．

命　　令	101	n
ディスプレースメント d16		
{目的オペランド（長さ 0, 1, 2)}		

d16は符号付き整数とみなされ，ディスプレースメントを示す．2の補数と

解釈するので $-32768 \leq d < 32768$. Anの内容を番地とみなし，それにd16を加えたもの，すなわち(An)+d16を実効アドレスとする命令である．

【例1】 MOVE　$100(A1)，$3000　　アドレス・レジスタA1の内容(たとえば $00001000)にd16=$0100を加える．すなわち

$$(A1)+d16=\$00001100$$

したがって $1100番地の内容を $3000番地に転送する命令である．

1.5.6 インデックス付きアドレス・レジスタ間接

任意の汎用レジスタRiをインデックス・レジスタに用いたアドレス・レジスタ間接である．命令形式は次の通りで長さは2～4ワードである．

命　令				110	n
D/A	i	W/L	000	ディスプレースメント d8	
{目的オペランド　(長さ 0, 1, 2)}					

W：ワード (0)
L：ロングワード (1)

第2ワードの先頭ビットが0ならRi=Di，1ならRi=Aiである，W/Lはインデックス・レジスタの長さを示すもので，0のとき16ビット，1のとき32ビットと約束する．ディスプレースメントd8は，1.5.5のd16と同じく符号付きの整数である．結局

$$(An)+(Ri)+d8$$

によりオペランドが決まる．

【例1】 MOVE　$04(A0, D0) $1000　　機械語で $31F00004 1000となる命令で，インデックス・レジスタはD0である．これを16ビットと見てオペランドを定める(もしMOVE　$04(A0, D0.L) $1000ならD0は32ビットのインデックス・レジスタになる)．

ここでA0に $2000, D0に $2BDCが入っていれば

$$\$2000+\$2BDC+\$0004=\$4BE0$$

となるので，$4BE0番地から $1000番地へ転送が行なわれる．

1.5.7 アブソリュート・ショート・アドレス

命令の長さは2～4ワードで次の形式をとる．次の§1.5.8とともに最もわかりやすい形式であろう．

命　　令	111000
実効アドレス	
{目的オペランド (長さ 0, 1, 2)}	

アブソリュート・ショートは 111000 という 6 ビットの 2 進数で示される

これはいわゆる2-アドレシング・モードそのものである.

【例1】 MOVE　$1000, $2000　　$1000 番地の内容を $2000 番地に転送する命令である.

1.5.8 アブソリュート・ロング・アドレス

命令の長さは 3~5 ワードで次のような形式をとる.

命　　令	111001
実効アドレス (2ワード)	
{目的オペランド (長さ 0, 1, 2)}	

アブソリュート・ロングは 111001 という 2 進数で示される

【例1】 NEG　$014000　　これは $14000 番地にある数の補数をそこにおく命令である. もし $14000 番地の内容が $0001 なら $FFFF になる. この場合, 目的オペランドはいらない.

【例2】 MOVE　$100000, $200000　　これは $100000 番地の内容を $200000 番地に転送する命令である.

1.5.9 イミディエート

操作の対象となるデータは命令部に続いており, 長さは 1~3 ワードである. 定数のデータを指定しているのであって, レジスタや, メモリ・アドレスを指定しているわけではない.

命　令　部	111100
{データ (長さ 1, 2)}	

もしデータがバイトなら, このデータは下位 8 ビットに入り, 上位 8 ビットは 0 になる.

【例1】 MOVE　#$1000, A0　　#はイミディエートを示し, データは $1000 なので 1 ワードをとる. これは 2 ワード命令で, A0 が $00001000 になる. この場合 A0 の符号は#で示される数の符号である. アドレス・レジスタに対しては, 符号拡張が行なわれ, データ・レジスタに対しては符号拡張が行

なわれない.

1.5.10　クイック・イミディエート

加減および転送だけに使われる. 定数のデータが8ビットしかない小さいときのみ有効な命令で, この場合のデータはロングワードとして符号拡張される.

転送の場合は

		8	7　　0
0111	レジスタ	0	データ

という形で与えられ, データ・レジスタDnへデータが転送される. この場合, データはロングワードとみなされて符号拡張されることに注意してほしい.

加減の場合は

15　　12	11　　9	8	7　　6	5	0
0101	データ	A/S	サイズ	モード	レジスタ
{目的オペランド (長さ0, 1, 2)}					

モードは000から111と7種類ある

という形で与えられ, 汎用レジスタまたは, メモリ・アドレスを利用できる. ただしサイズがバイトのときは, アドレス・レジスタ直接は利用できない. 目的オペランドがレジスタのときは, 1ワード命令になる. サイズは目的オペランドの長さを指定し, バイトとワードとロング・ワードに分かれている.

第1ワードの第8ビットは加算のとき0, 減算のとき1である. 加減されるデータは1から8までである(0が8を表わす).

【例1】 ADDQ . L#1, A0　ロングワードと見てアドレス・レジスタA0に1を加算する命令である.

【例2】 MOVEQ　#$56, D3　データ・レジスタD3に$00000056を転送する命令である. したがって実行後D3の内容は$00000056になる.

1.5.11　相対アドレス

命令のある場所からの距離を与える方法によって実効アドレスを決めようという考えで, この距離はディスプレースメントdのある番地を中心として−32768から32767に及ぶ. 命令の形式は次の通りで, 長さは2〜4ワードである.

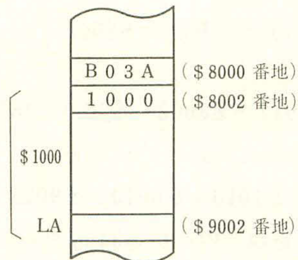
命　　令	111010
ディスプレースメント d16	
{目的オペランド (長さ 0, 1, 2)}	

目的オペランドが汎用レジスタなら，この情報は命令部にのり，命令全体の長さは2ワードになる．プログラム・カウンタ(PC)の内容は，この命令の第1ワードを読み取った時点では†，第2ワードを指している．そこで(PC)+d16という演算を行なって実効アドレスが決まる．

【例1】 MOVE .W<LA>, D0　　このLAはラベルで，もし実効アドレスが$9002番地を指していれば，ディスプレースメントは

$$\$9002-\$8002=\$1000\ (=4096)$$

となる．すなわち $9002番地の内容がデータ・レジスタD0に転送される命令である．



1.5.12 インデックス付き相対アドレス

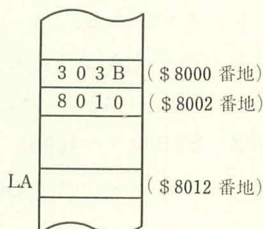
ここでは，ディスプレースメントdは8ビットの符号付き整数で，§1.5.11の方法にさらにインデックス・レジスタRiの内容を加えようというものである．したがって命令形式は次の通りで，長さは2~4ワードである．

命　　令				111011
D/A	i	W/L	000	ディスプレースメント d8
{目的オペランド (長さ 0, 1, 2)}				

† 実は命令に対してプリフェッチ(あらかじめ次の命令全体を読み出して速く動作する)を行なっているので，正確にはこうなっていない．しかしここではこう理解したほうがわかりやすい．

第2ワードの先頭ビットでデータ・レジスタ(0)かアドレス・レジスタ(1)かを定め，次にレジスタ番号を入れる．また，次のW/Lで，インデックス・レジスタRiが16ビット(ワードW, 0)か32ビット(ロングワードL, 1)かを定める．ディスプレースメントd8は符号付き整数と見て32ビットに拡張され，(PC)＋(Ri)＋d8を実効アドレスとする(−128≦d8<128)．

【例1】 MOVE　<LA> (A0), D0　　この命令が$8000番地にあると



すれば，命令を読み終わった時点で(PC)＝$8002となっている．もしLAが$8012番地を指していれば

$$d8 = \$8012 - \$8002 = \$10\ (=16)$$

このとき(A0)＝$1010とすれば実効アドレスは

$$\$8002 + \$1010 + \$0010 = \$9022$$

番地になる．すなわち$9022番地の内容が$3456なら，D0にこの$3456が転送される．ここではインデックス・レジスタはA0である．

1.5.13 インプライド

ステータス・レジスタやプログラム・カウンタなどを指定対象とする命令をインプリシット命令といい，そのアドレス方法をインプライド・アドレシングという．対象としてはステータス・レジスタ(SR)，条件レジスタ(CCR)，プログラム・カウンタ(PC)，システム・スタック・ポインタ(SP)，ユーザ・スタック・ポインタ(USP)，スーパバイザ・スタック・ポインタ(SSP)がある．

プログラム・カウンタが含まれているので，各種テストやブランチはインプリシット命令に含まれている．命令形式は次の通りで，長さは1〜3ワードになる．

命　　令	モード
{目的オペランド（長さ 0, 1, 2)}	

ここのモードや目的オペランドは単純でない.

【例1】 BNE NEXT　　ステータス・レジスタのZフラッグの内容が0ならNEXTというラベルの付いたアドレスに分岐せよという命令で，これは 66 | d8 という形をとり，d8は8ビットのディスプレースメントである．もしロング・ブランチしたければ，次の形を用いればよい.

6600
d16

以上で命令のアドレス・モードについて概説がすんだ．これとアセンブラ命令をうまく使えば面白いプログラムが作れる．多くの命令は，今まで見てきたように2-オペランド方式を用いており，68000の柔軟性および能力が大幅に高められている.

入出力はメモリ・マップ方式によっている．すなわち周辺機器に対し（複数個の）メモリ・アドレスが与えられ，このアドレスを指定することが入出力につながる．したがって入出力関係の特殊命令はもっていない.

参考として表1.8に68000の命令コードをあげる.

表 1.8　68000の命令コード

ビット 15-12	命　　令	ビット 15-12	命　　令
0000	ビット処理，MOVEP，イミディエート	1001	減算，拡張付き減算
0001	バイト転送	1010	使用していない(例外付き割込み，1010エミュレータ(㈱日立製作所))
0010	ロング・ワード転送	1011	比較，排他的論理和
0011	ワード転送	1100	論理積，乗算，拡張付き10進加算，交換
0100	いろいろな命令，その他		
0101	クイック加減，条件テスト関係	1101	加算，拡張付き加算
0110	ブランチ	1110	シフト，ローテイト
0111	クイック転送	1111	使用していない(例外付き割込み，1111エミュレータ(㈱日立製作所))
1000	論理和，除算，拡張10進減算		

1010，1111はMotorola社では使っていない

1.6 ソフトウェアの展望

68000ファミリの日本におけるセカンド・ソースは(株)日立製作所で，その周辺LSIを含めて日立武蔵工場から出荷されている．後続の68008，68010，68020やその周辺LSIも当然それに続くことになろう．

すでに同社では68000ファミリのいろいろなソフトウェアを開発しており，これは後章で詳しく説明される．ここではMotorola社と(株)日立製作所のソフトウェアを中心として展望しよう．

1.6.1 Motorola社の基本ソフトウェア

Motorola社で作られた基本的なオペレーティング・システムはVDOSとそれを含んだVERSA dosである．これはマルチ・ユーザ/マルチタスク・システムで，リアルタイム/オンラインに使え，制御その他には大変有効である．すでに開発支援装置EXOR macsその他で用いられ，構造的プログラミングにむいたマイクロ・アセンブラやPascalコンパイラを使うことができる．コ

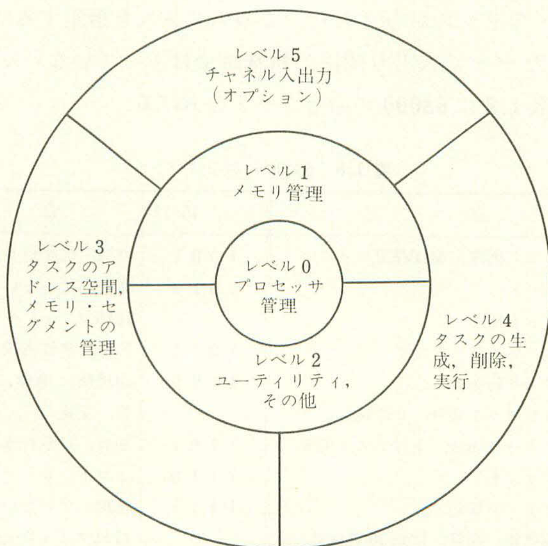


図 1.20　RMS68Kの展望

ンパイラとしてはこのほかにFORTRAN, BASIC-M, COBOL, Ada, APL, FORTH, MPL (PL/I類似のもの) などがある．もちろんこれらによって作られたプログラムのリンクやデバッグが可能である．

VDOSはVERSA dosとほぼ同様な製品で，入出力の管理その他を行ない，その中にRMS68K (real-time multitasking software) を含んでいる．これはプロセッサ，メモリ，タスクの管理などを行なう部分である．

RMS68Kは六つのレベルに分かれる(図1.20)．

レベル0：プロセッサを管理する

レベル1：物理的メモリを管理する

レベル2：ユーティリティ機能に関する部分

レベル3：タスクのアドレス空間とメモリ・セグメントを管理する

レベル4：タスクの生成実行をつかさどる

レベル5：チャネル用入出力機構を管理する(オプション)

レベル0～4はプログラム格納のための12Kバイト，レベル5は4.5Kバイトのメモリを要するが，これは最大必要量であって用途に応じて小さくできる．今日ではRAMやROMが低価格で入手できるから，この領域は決して大きいものではない．

このほか，言語CとUNIXを塔載する計画が進行している．CはPDP-11/70上のUNIXオペレーティング・システムのためのプログラム言語として開発されたもので，PDP/11とよく似たアーキテクチャをもつ68000には容易に塔載できる．

なお，Adaを68020によるシステムの中心言語とするようである．

1.6.2 基本ソフトウェア

(株)日立製作所ではRMS (real time monitor system) を上記のものとは別個に開発しており，タスクの管理などは容易に行なえる．これはMotorola社のRMS68Kとは異なるが機能はよく似ている．すなわち多重処理を行ない，タスクの変更や連絡(メッセージの送受信)などを行なう．メモリはMotorola社よりも小さい．これについては第3章を見られたい(表1.9)．

同社はDigital Research社のCP/M-68K開発に協力しており，CP/M-86やCP/M-80で作られた応用ソフトウェアをそのまま，または多少修正して68000システムで使うことができる．このため少なくとも128Kバイトのメモ

りと（フロッピーまたはハード）ディスクおよびコンソールが必要である．もちろんプリンタ，カセット，紙テープやカードなどの機器を含む周辺装置を容易に接続することができる．

ソフトウェアはBIOS (basic I/O system), BDOS (basic disk operating

表 1.9　RMSの機能一覧表　　HD68000用RMSの機能は，S680RMS3Rの仕様に基づいている

品　種			HD6800用 RMS	HD6809用 RMS	HD68000用 RMS
提供媒体および容量			マスクROM, 4Kバイト	マスクROM, 8Kバイト	EPROM, 40Kバイト
タスク管理	最大タスク数		32	256	255
	タスク制御		・起動・停止 ・POST/WAIT	・起動・停止 ・POST/WAIT ・ENQ/DEQ ・禁止・解除	・起動・停止 ・POST/WAIT ・ENQ/DEQ ・禁止・解除 ・中断・解除 ・メッセージ交換 ほか
	排他制御資源数		—	255	255
	タイマ	タイマ数	255	255	255
		最小単位	10 ms	10 ms	1 ms
入出力管理	入出力装置数		64	255	255
	標準入出力装置		・コンソール (RS232Cレベル TTLレベル) ・電子式卓上計算機タイプコンソール	・コンソール (RS232Cレベル TTLレベル) ・電子式卓上計算機タイプコンソール	・コンソール (RS232Cレベル TTLレベル) ・プリンタ（セントロインタフェース） ・フロッピーディスク
デバッグ機能	コマンド数		9	19	23
	デバッグ機能		・タスク制御 ・タスク状態表示 ・メモリダンプ ・メモリ変更 ほか	・タスク制御 ・タスク状態表示 ・メモリダンプ ・メモリ変更 ・ブレークポイント ・事象トレース ほか	・タスク制御 ・タスク状態表示 ・メモリダンプ ・メモリ変更 ・ブレークポイント ・事象トレース ・プログラムロード ・メモリ転送 ほか
ファイル管理機能			オプション	計画中	標準

（(株)日立製作所の提供）

system), CCP (console command processor) およびTPA (transient program area) で構成されており, 図1.21のように関連している.

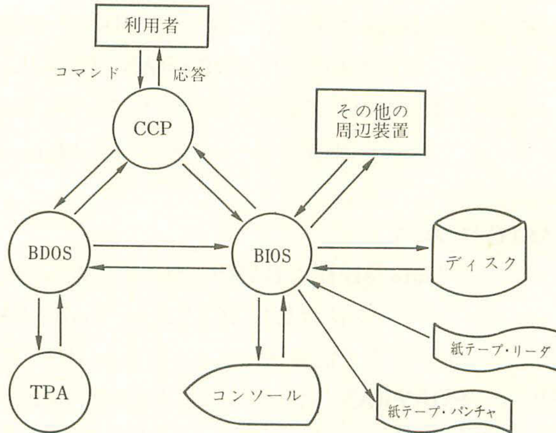


図 1.21 CP/Mの構成

BIOSはディスクやコンソールその他の周辺装置との基本的なソフトウェア・インタフェースに相当する部分で, ハードウェアを変更するときはこの部分の修正が必要である(現在の大きさは2Kバイト).

BDOSはディスクを管理する部分で, ファイルに関係し, 現在の大きさは3.5Kバイトである.

CCPはコンソールからのコマンドを解読し, そのための処理を行なう場所で, 主メモリに常駐し約2Kバイトを要する.

TPAは, CCPによってディスクからロードされたプログラムを格納するエリアで, ユーザ・プログラムはここにロードされて実行される.

コマンドなどは従来のCP/Mの拡張になっているので大変使いやすい. 開発はネットワークを最終目標としてconcurrent CP/M-68K, MP/M-68K, MP/NET-68Kの順序で行なわれる.

CP/M-68Kはシングル・ユーザ/シングル・タスク方式の簡易版CP/MでCP/M2.2に相当している. concurrent CP/M-68Kはシングル・ユーザ/マルチ・タスク方式で標準のCP/Mといえる. MP/M-68Kはマルチ・ユーザ/マルチ・タスク方式を, MP/NET-68Kはネットワーク方式を目標としている.

これによって8ビット・マイコンむけに作られた流通ソフトが68000でも使えるようになる.

このほか，(株)日立製作所でもフロッピー・ディスク，ハード・ディスクをもったシステム開発装置 SD300 を発売しており，FDOS を搭載している．こういうオペレーティング・システムの下に FORTRAN, Pascal, S-PL/H (PL/I 相当)† などが使えるようになっており，応用プログラムの開発・作成は大変やさしくなっている．こういったコンパイラについても Motorola 社と (株)日立製作所は協力している.

1.6.3 BASIC について

BASIC については Motorola 社の BASIC-M，安立電気(株)の BASIC，横河・ヒューレット・パッカード社の BASIC, Tandy Radio Shack 社のモデル 16 の BASIC などがある．安立電気(株)ではこのほか UCSD の Pascal も使え，これを利用して FORTRAN も使える．この UCSD の Pascal, FORTRAN 77 は Apollo 社の DOMAIN でも使える．これはその名の通りネットワークによって多数の DOMAIN ワークステーションをつなぐことができるシステムである.

1.6.4 Pascal, S-PL/H, FORTRAN について

いくつかのコンパイラについてまとめておこう.

Pascal は，チューリッヒ大学の N. Wirth 教授により設計された言語から始まり，UCSD 仕様の標準 Pascal にいろいろな仕様を加え，大変強力になっている．マイクロコンピュータ用の tiny Pascal も開発されたが最近はあまり使われていない.

Motorola 社では 68000 用の Pascal を開発し，安立電気(株)では UCSD の Pascal を使用している．どちらも機能としてはよく似ており，プログラムの系統的設計およびアルゴリズムの記述には大変便利である．ALGOL の進化したものと考えてもよく，プログラムは begin と end で囲まれたブロック構造を主体としている．ファイルの編成や入出力などについては ALGOL よりも使いやすいかも知れない．UCSD p-system として，Pascal (version 4.1), FORTRAN，などが使えるほか，ネイティブ・コード・ジェネレータもあり，実行速度を上げることもできる．また，原始プログラムを変換する処理系が

† スーパ・ピーエルエイチと読む．super-PL/H の略.

COBOL, FORTRANなどのコンパイラに比べて小さく, 移植性が高いことも大きな特徴の一つである.

(株)日立製作所のS-PL/HとFORTRANについてまとめておこう.

現在のところ, いろいろなプログラム言語が使われているが, その基準としては

(1) 標準的言語で誰にでも理解できること

(2) 一貫したソフトウェア体系を作っていること

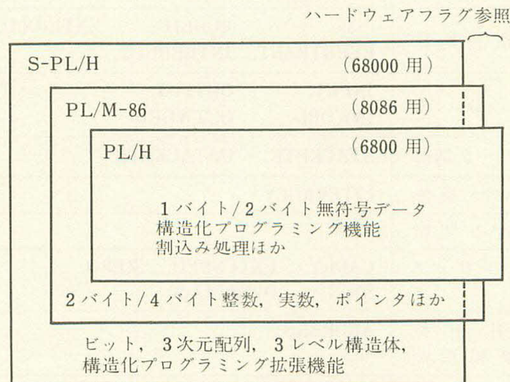
が重要である. JIS規格やISO規格がある場合は, それを含み, 上記の2点を中心としてその仕様を決定する必要がある.

PL/Iに相当するS-PL/Hは, (株)日立製作所で開発された6800用のPL/HおよびIntel社のPL/M-86を含み, リロケータブル・オブジェクト・プログラムを直接生成するレジデント・コンパイラ方式と, 大型計算機などを用いてオブジェクト・プログラムを生成するクロス・コンパイラ方式がある(図1.22). 最適化を行なっているので大変能率のよいオブジェクト・プログラムを作成することができる. この最適化により, アセンブラによるものと比べて, 150%程度の大きさになるといわれている. 主な特徴は次の通りである.

(1) ビット形データを扱うことができる

(2) 並列制御用のTEST AND SET文が追加された

(3) データの個々の値に対応する場合分け処理が単純に書ける選択肢付き



((株)日立製作所の提供)

図1.22 S-PL/H言語の位置付け

CASE 文がある

(4) 繰返し処理を途中で打ち切る EXIT がある

表 1.10 にその仕様を示す.

表 1.10 S-PL/H 言語機能　S-PL/H は数値や文字ばかりでなく，ビットデータや割込みに対する処理機能も備えた，構造化プログラミングに適するシステムプログラムむき言語である.

項目		仕様
データ要素の型		INTEGER, LONG INTEGER, BYTE, WORD, REAL, BIT, POINTER
変数属性		3次元配列, STRUCTURE, BASED, INITIAL, DATA, AT, PUBLIC, EXTERNAL
演算子		+, −, *, /, MOD, =, < >, <, <=, =>, >, OR, AND, XOR, NOT, PLUS, MINUS, @
基本文	代入文	単純, 多重, 埋込み
	制御文	CALL, RETURN, GO TO, EXIT, 空文
	機種依存文	ENABLE, DISABLE, HALT, TEST AND SET
条件文		IF THEN ELSE
DO ブロック		DO WHILE, 反復 DO, DO CASE, 単純 DO
手続き属性		内部, PUBLIC, EXTERNAL, REENTRANT, INTERRUPT
入出力		INPUT, OUTPUT, INWORD, OUTWORD
スタック制御		STACKPTR, USTACKPTR
マクロ機能		LITERALLY
組込み関数		43種
ハードウェアフラグ		CARRY, EXTENDED, ZERO, SIGN, OVERFLOW
PL/H 上位互換用機能		ADDRESS, −>
省略形予約語		ADDR, DCL, EXT, INIT, LIT, PROC, STRCT

((株)日立製作所の提供)

FORTRANはANSI FORTRAN 77†の標準サブセットを含み，ビット処理機能††を追加したもので，レジデントとして使うことができる(図1.23).

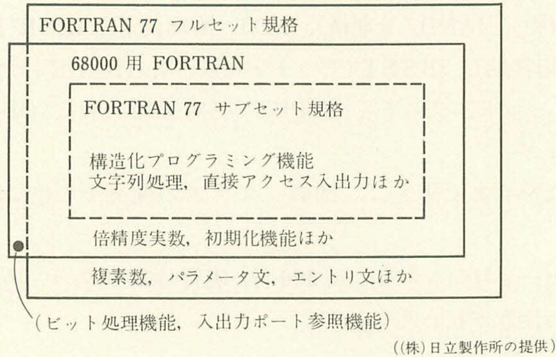


図 1.23 HD68000用FORTRANの位置付け

実数の規格としては32ビットの単精度および64ビットの倍精度があり，IEEE規格に従っている(図1.24).

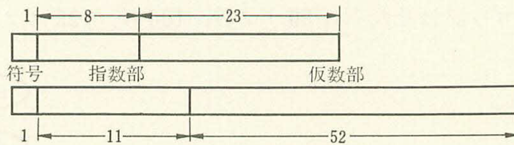


図 1.24

このFORTRANは次の特徴をもっている.

(1) ブロックIF文があり，構造化プログラミングが容易である(IF～THEN～ELSEおよびENDIF)

(2) 文字型変数および配列が使える

(3) メモリ内のデータをファイルとして取り扱うことができる

(4) 副プログラムの変数および共通ブロックの内容がRETURNまたはENDの実行後も保持できる(SAVE文)

整数としては16ビットの短整数および32ビットの標準がある．もちろん混

† 厳密にはANSI X3.9-1978 FORTRAN 77という.

†† bit manipulation, ISA-S61.1 .

合演算が可能なので，変数の型を意識しないでプログラムを作ることができる．

ビット処理機能としては

IOR（論理和），IAND（論理積），NOT（論理否定），IEOR（排他的論理和），ISHFT（桁移動），IBSET（ビット・セット），IBCLR（ビット・クリア），BTEST（ビット・テスト），INPUT（絶対番地入力），OUTPUT（絶対番地出力）など

がある．これをうまく使えば，制御システムの開発その他には大変有効であろう．

診断メッセージはコンパイル時300種，実行時28種あり，誤りの発見その他にはかなりの注意が払われている．

1.7　周　辺　LSI

現在68000には12.5 MHzを基本周波数とするL12を頂点として，L10，L8，L6，L4があり，その基本周波数はそれぞれ10MHz，8MHz，6MHz，4Mzである．そのクロックはそれぞれ80ナノ秒，100秒，125ナノ秒，167ナノ秒，

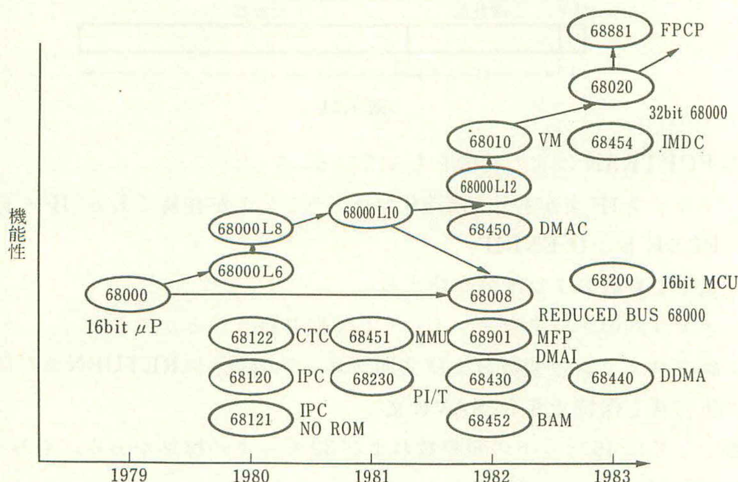


図 1.25　68000と周辺LSI相関図

表 1.11 68000 の周辺 LSI

LSI 番号	略　称	主たる用途その他
68120	IPC	intelligent peripheral controller. 6800 のような 8 ビット CPU と 68000 を結ぶ 48 ピンの入出力コントローラ
68230	PI/T	parallel interface/timer. 単方向または双方向で用いるパラレル I/O 用のインタフェース. 8 ビットまたは 16 ビットの bit I/O ができる. 24ビット・プログラマブル・タイマを含み, タイマによる割込みも可能である
68440	DDMA	dual direct memory access controller. DMAC をデュアルにしたもの
68450	DMAC	direct memory access controller. 周辺機器と主メモリの間のデータ転送を 4 M バイト/秒で行なう能力をもつ. 4 チャネル処理ができる
68451	MMU	memory management unit. 主メモリのアドレス変換とメモリ保護を行なう 64 ピンのユニット. ページングとセグメント処理をサポートする. オペレーティング・システムのオーバヘッドを減少させる効果がある
68452	BAM	bus arbitration module. 多数の CPU を使うシステムで, どのバス・マスタがバスを使ってよいかを決めるアービトレータ
68540	EDCC	error detection and corrention circuit. データのチェックと訂正を行なう 48 ピンの LSI. バイトでもワードでもよい. 誤りがあるとエラー・フラッグを立てる
68561	MPCC-II	multi-protocole communication controller. シリアル・データ通信用インタフェースで, 転送は同期でも非同期でもよい. 全二重または半二重で使う. 水晶発振器とボーレート生成器を内蔵する
68652	MPCC	multi-protocole-communication-controller. 同期型シリアル・データ通信用インタフェース. BOP, BCP (bit-oriented protocole, byte-controle-protocole) の双方に使える
68653	PGC	polynomial generator and checker. データ送受信で双方のキャラクタの一致を確めるための 16 ピンの LSI. 6, 7, 8 ビットのキャラクタをチェックする
68661	EPCI	enhanced programmable communication interface, データ通信用 LSI で 16 種類のボーレートを設定できる. マイクロプロセッサからデータを受け取り, パラレルに変換して送り出す. 受取りも同様
68881	FPCP	floating-point coprocessor. 浮動小数演算を行なう LSI で, 68020 のときはコプロセッサとなる. 四則演算, 平方根などの計算を行なう. 46, 47 ページ参照

250ナノ秒である.

たとえば，最も速い転送命令は4クロック・サイクルで実行されるので，L8なら500ナノ秒，L12なら320ナノ秒で完了する．しかし，この68000といえども，これだけでは何もできない．周辺機器と68000を結び付ける周辺LSIが必要である．図1.25に68000(マイクロプロセッサ)と主な周辺LSI(バスマスタ・コントローラなど)との相関図を示す．この周辺LSIは次々と開発または改良されつつあるが，知り得た範囲でそれを表1.11に示した.

たとえば，68450はDMA制御用LSIすなわち「レジスタを経由しないでデータを直接アクセスするためのLSI」である．これによりディスクと主メモリとの間で直接にデータをやり取りできる．なお，これらのLSIはMotorola社，Mostek社，Signetics社，Rockwell社，(株)日立製作所などが分担して開発し，複数メーカが生産している.

表1.10の内容の一部はMotorola社の*Advance Information*や*Product Preview*によった．その意味で今後多少変更があるかもしれない.

68881は浮動小数用の80ビット・レジスタを8個有し，IEEEの標準に従って演算を行なう．そのデータの形式は

s	e	m

となっており，sは符号ビット，eは指数部，mは仮数部を示す．そのビット数は表1.12の通りである.

表 1.12

	単精度	倍精度	高精度
符号部s	1	1	1
指数部e	8	11	15
仮数部m	23	52	64
合計	32	64	80

整数やBCDは自動的に80ビットの数としてレジスタに入ってしまうので，混合モードで演算ができる．機能は次に見るように実に豊富である．

(1) 加減乗除

(2) 剰余

(3) 比較
(4) 平方根
(5) 整数化
(6) 丸め（四通りの丸めが可能である：最も近い整数への丸め，大きい（小さい）方への丸め，絶対値で0に近い方への丸め）
(7) 実数化（単精度，倍精度，高精度）
(8) 正または負の無限大，符号なしの無限大
(9) 絶対値
(10) 誤りの検出
(11) 正弦，余弦
(12) 逆正接
(13) 2またはeを底とする対数
(14) eの累乗

このほか，正接，双曲線関数，常用対数，10の累乗，一般の累乗 y^x も予定されており，これらはいずれも高精度で計算される．はっきりしたことはでき上がってからのお楽しみで，確定していない部分もある．

この68881を68020に接続する場合は，68000で使われなかった命令を用いて使うようになっている．この意味でコプロセッサと呼ぶ．この68881により，コンパイラの作成などは大変楽になろう．

1.8 68000に続くプロセッサ

Motorola社は68000に続いて68008，68010，68020を発表した．その概要を紹介する．

1.8.1 68008

これは8ビット・データバスを備えた68000といえるもので，レジスタについては68000と同じである．ただしプログラム・カウンタは20ビットなので，主メモリは1Mバイトとなっている．外部に出ているアドレス線は20本で，全体が48ピン・パッケージに納められ，68000よりコンパクトである（図1.26）．

ステータス・レジスタは68000と同じで，アドレス方式も14種あり，命令は全く同じである．データ・バスの幅が68000の半分なので，68000と比べる

と（同一クロック周波数で）60％の能力をもつといわれている．しかし16/32ビット・マイクロプロセッサの縮小版なので，いわゆる8ビット・マイクロプロセッサより優れていよう．周辺LSIは6800用のものと互換性をもっており，6800や6809よりも使いやすい．

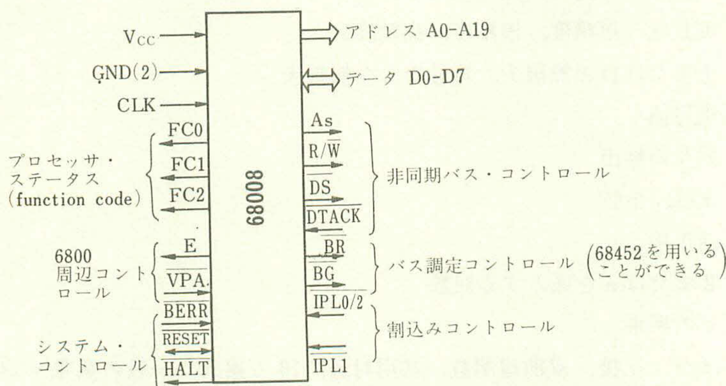


図 1.26　68008 のシグナル・ダイヤグラム

主として8ビット・データを使う分野にむいている．したがって8ビット・マシンの分野で高速処理を要する場合には特に有効である．

6800周辺コントロールは同期用インタフェースで，6800の周辺LSIを接続するのに使われる．このほかに非同期バス・コントロールがあり，周辺機器の制御に関しては柔軟な対応が可能である．

これらの点を総合すると，68008は68000とは異なる分野，たとえば文字データの高速処理が必要なシステムや従来の8ビット・マイクロプロセッサではあきたらない場合などで，今後の成長が期待できるマイクロプロセッサである†．

1.8.2　68010

これは68000を仮想システム用にしたマイクロプロセッサである．68000では仮想記憶システムを実現する際，多少問題であった．

仮想記憶システムでは，命令の対象となる情報が主記憶にない場合，ただちにその実行を中断し，ディスクなどから必要な情報を得て処理を再開する．こ

† たとえば入試のコンピュータ処理やワード・プロセッサなどにはむいていると思う．

のためにはプロセッサの内部状態を（再開されるまで）どこかに保存しておかねばならない．こういう機能は68000にはなかった．このためApollo社のDOMAINのように二つの68000を用いて仮想記憶システムを実現した例がある（必要な場合には一方で情報を確保する）．

このため，68010ではスタックに内部状態を積み，最後にRTE命令†を実行して内部状態を再現できるようにした．

68010では1024バイトの割込みベクタ・テーブルが再配置可能なので，特に0番地からと限定する必要がなく，複数個のオペレーティング・システムを走らせることができる．このため32ビットのベクタ・ベース・レジスタVBRが新設された．

このほか，MOVES（move to alternate address space）とMOVEC（move controle register）という命令が新設されている．これはオペレーティング・システムの下に動作しているときだけ実行できる命令で，ファンクション・コード変更用レジスタ（alternate function code register）に関係がある．このレジスタはソースsとディスティネーションdとから成り，4ビット×2で構成されている．これにMOVEC命令で変更すべきファンクション・コードを書き込んでおき，MOVESで出力する．これによってオペレーティング・システムとユーザの間，またはユーザ同士の間でのデータの受渡しができるようになった．

算術演算や論理演算などの命令のクロック・サイクルが2サイクル短くなり，性能が68000よりも25％程度向上している．また，クリアCLRなども速く実行できるようになった．

誤り検出回路の設計が以前より容易となっているが，その詳細は省略する．

1.8.3 68020

実際には，この68020を基本として68000が実現されたといわれている．68020のプログラム・カウンタは32ビットで，それを完全に利用できるので4Gバイトのアドレス空間を設定することが可能である．

アドレス方式が40種以上にふえ，命令体系も拡張され，キャッシュ・メモリで命令を実行するようにしてある．したがって，キャッシュ・メモリ内での

† return from exception　例外からの復帰命令．ステータス・レジスタとプログラム・カウンタが復帰する．

小さなループは，そのままで回るので，実行速度は向上し，これに接続したプロセッサやDMAコントローラなどが働きやすくなっている．しかも浮動小数用コプロセッサ68881を接続でき，関数計算もやれるので，コンパイラ設計も容易である(68881は68000の場合には周辺チップとして接続される)．これについてはもっと詳しいレポートが出るまで解説を控える．

なお，図1.25に16/32ビット・マイクロプロセッサと主な周辺LSIとの相関を示してあるので，参照してほしい．

1.9 VERSA module その他

1.9.1 VERSA module と VME module

68000のバスに関する命令は次のように分類される．

(1) データ転送に関するもの：アドレス・バス，データ・バス，制御信号が関係する

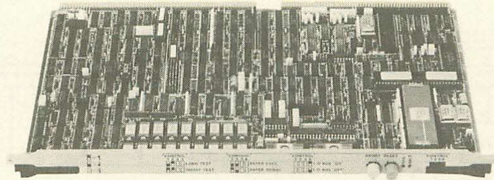
(2) バス調停に関するもの：バスのマスタ・シップ要求，承認，確認が関係する

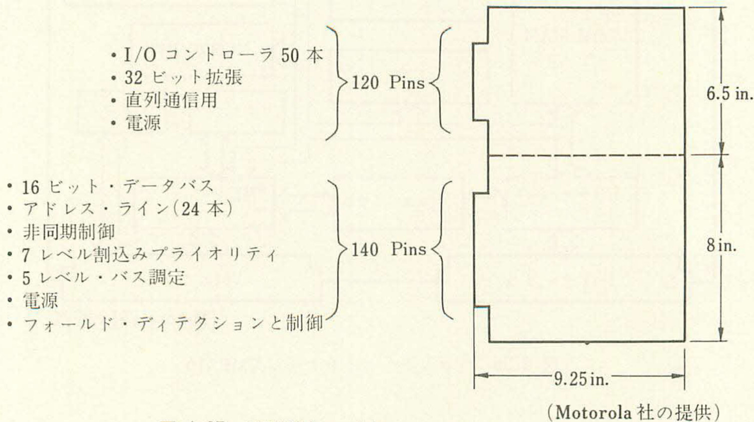
(3) バス・エラーと停止に関するもの：外部デバイスとのハンドシェイキングに関するエラーと停止が関係する

(4) リセットに関するもの：バス・エラーや再起動に関するバス・サイクルの制御が関係する

ここでは，これらをきちんと説明しようというのではない．こういういくつもの命令により，外部デバイスを制御しているということ，そのためには68000に周辺LSIを付け，周辺機器とのインタフェースを考えることなどが必要であることを知っておかねばならない．

このため，コンピュータ・システムを構成する場合，このバスに関する命令などがうまく伝わるようにモジュールをまとめなければならない．このモジュールとして以前はS-100が使われた(これはZ80その他の8ビット・マイクロプロセッサの標準規格である)．Motorola社ではVERSA moduleを作って利用者の要求に応えている．これは235×368.3mmの大きさのもので，50ピンの入出力用端子を含む120ピンとデータ・バスを含む140ピンのVERSA busを1枚のボードにまとめたものである．図1.27に見るようにかなり大きい．





(Motorola 社の提供)

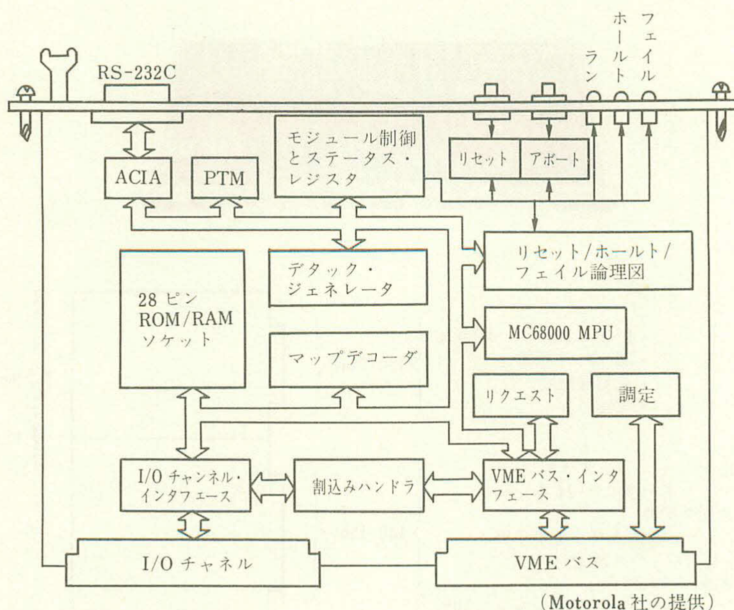
図 1.27 VERSA module

もちろん 68020 を意識して作られた 32 ビット・バス・システムである．これを用いた VMC 68/2 というコンピュータ・システムが最近発売された(序文参照)．

さらに最近 Mostek 社，Signetics 社と協力して Euro Card というモジュールが発表された．Motorola 社ではこれに VME module という名を付け，VME 110-1 などの形で出している．そのブロック・ダイアグラムを図 1.28 に示す．

これは 160×233.4 mm の大きさをもち，VERSA module よりは小さい．もちろん 16 M バイトまでアドレスできるように，96 ピンの入出力端子とバス用の端子をもっている．たとえば VME110 はシリアル，パラレル両方の入出力が可能で，パラレルはセントロニクスでもよい．このほか GP-IB なども可能である．

もちろん 68008，68010，68020 を使えるように考えられており，システムのグレード・アップも十分意識して設計されている．



（Motorola 社の提供）

図 1.28　ブロック・ダイヤグラム VME110

1.9.2　そ　の　他

68000の新規採用数の状況を図1.29に示す．ここに示す通り，68000はかなりよく使われるようになった．（1983年には68000を使ったシステムが非常に増えた．）

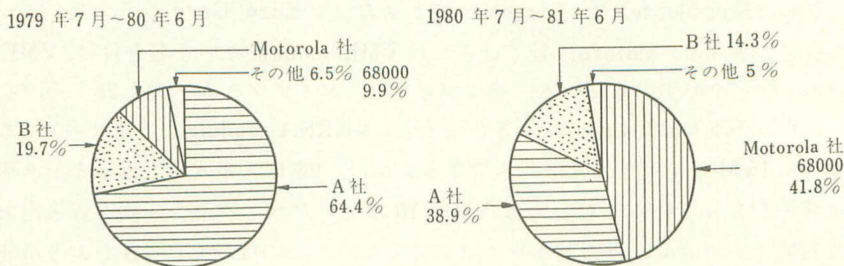


図 1.29　16ビット・マイクロプロセッサ新規採用数比率　（Motorola 社の提供）

その資料によれば16/32ビット・マイクロプロセッサとその周辺LSIの市場

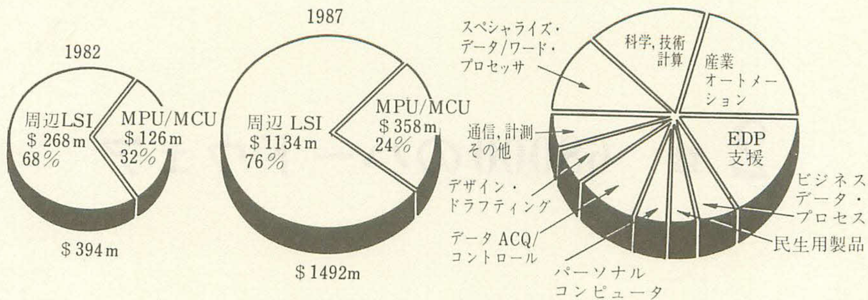


図 1.30 市場予測と応用分野（Motorola 社の提供）

は 1987 年には約 15 億ドル（約 3 500 億円）に達し，年率でプロセッサが 23 %，周辺 LSI で 34 % の成長が予想されている（図 1.30）．

こういう意味で，マイクロプロセッサは深く広く使われるようになり，言語もアセンブラだけではなく，FORTRAN や COBOL，PL/I などの高級言語が使えるようになった．

いずれは 32 ビットのマイクロプロセッサの種類もふえるものと思われるが，それでも 68000 は十分意味のあるマイクロプロセッサである．この次のマイクロプロセッサはどんなアーキテクチャをもつであろうか．これは大変興味のある難しい問題である．筆者は二つの方向に分化するものと予想している．すなわち，一つは 4 ビット，8 ビットに代表されるマイクロプロセッサであり，もう一つは 32 ビットに代表され，大型コンピュータを十分意識したシステムの中心プロセッサである．現在は 32 ビットがやっと注目されるようになったところであるが，高精度計算を十分意識した 80 ビットの時代が来るかもしれない．

2章　68000のハードウェア

68000は，Digital Equipment Corporation（DEC）社の有名な16ビット・ミニコンピュータ PDP-11のアーキテクチャをベースに，最新の半導体技術，回路設計技術から生まれた本格的16ビット・マイクロプロセッサである．

68000の内部アーキテクチャの特徴は，何といっても32ビット長の各種レジスタにあり，将来の32ビット・マイクロプロセッサへの拡張を十分に考慮した，進歩したアーキテクチャをもつマイクロプロセッサということができる．

本章では，68000のハードウェアを概説する．

2.1　68000 MPUの概要

68000 MPU (micro processing unit) は，64ピン・デュアル・インライン・パッケージに収められており（図2.1），1983年現在，日本では，(株) 日立製作所から4 MHz (HD 68000-4)，6 MHz (HD 68000-6)，8 MHz (HD 68000-8)，12.5 MHz (HD 68000-12) の各バージョンが供給されている．

68000の特徴としては，次のようなものがある．

(1)　豊富な32ビット汎用レジスタ

汎用レジスタ17個（32ビット長）：データ・レジスタ8個，アドレス・レジスタ9個（うち2個はスタック・ポインタ）

プログラム・カウンタ1個（32ビット長）

ステータス・レジスタ1個（16ビット長）

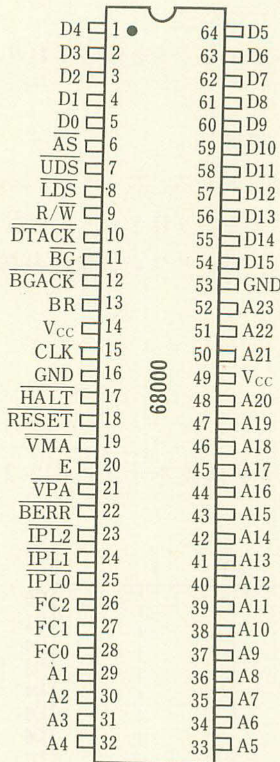


図 2.1 68000 のピン配置

(2) 16M バイトのアドレス空間

23 ビットのアドレス・バスによって，セグメントに分けることなく 16M バイトのメモリ・アドレスをダイレクトに指定することができる．

(3) 強力な命令セット

56 種の基本命令がある．符号付きおよび符号なし乗除算命令や，オペレーティング・システム，高級言語，構造化プログラミングなどをサポートする命令もその中に含まれている．

(4) 5 種類のデータ・タイプ

ビット，BCD (binary coded decimal) データ (4 ビット)，バイト・データ (8 ビット)，ワード・データ (16 ビット)，ロング・ワード・データ (32 ビット) の五つのデータ・タイプをサポートする．

(5)　豊富なアドレシング・モード

6形式14種類のアドレシング・モードを備えており，柔軟なアドレス指定が可能である．

(6)　多様なエクセプション処理†

リセット，バス・エラー，アドレス・エラー，トレース，7レベルの割込み，不当命令，特権違反，ゼロによる割り算といった合計256個のエクセプション処理ベクタが準備されており，使いやすい信頼性の高いシステムを構成することができる．

2.2　レジスタの構成

図2.2に，68000のレジスタ構成を示す．図2.2に示すように，68000は

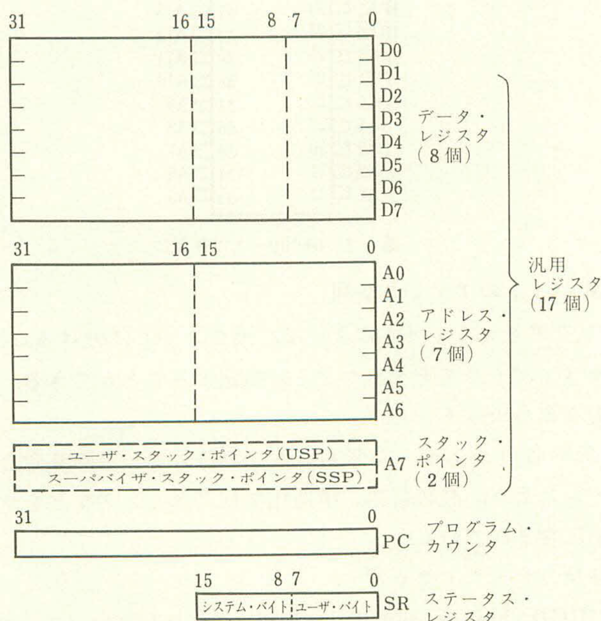


図 2.2　68000のレジスタ構成

† 例外処理ともいう．

32ビットの汎用レジスタを17個，32ビットのプログラム・カウンタを1個，それに16ビットのステータス・レジスタを1個備えている．17個の汎用レジスタはデータ・レジスタ8個，アドレス・レジスタ7個，スタック・ポインタ2個(うち1個がユーザ・プログラム用で別の1個はスーパバイザ用)から構成されており，汎用レジスタ17個はすべてインデックス・レジスタとして使用することもできる．

2.2.1 データ・レジスタ(D0～D7)

データ・レジスタは，8個すべてをバイト(下位8ビット)，ワード(下位16ビット)およびロング・ワード(32ビット)のデータの演算用レジスタとして使用することができる．命令の処理の対象となるデータの行き先アドレスをデスティネーションというが，データ・レジスタをデスティネーション・オペランドとして使用した場合，演算の対象となる下位のバイトまたはワード部分のみが変化し，残りの上位部分は変化せずにそのまま保存される．

2.2.2 アドレス・レジスタ(A0～A6)

アドレス・レジスタは，主にデータやジャンプ先のアドレスを指定するベース・アドレス・レジスタとして使用される．また，ワードおよびロング・ワード(バイトを除く)のアドレス演算用のレジスタとしても使われる．アドレス・レジスタをデスティネーション・オペランドとして使用した場合，オペレーションのサイズ（ワード）とは無関係にアドレス・レジスタ全体が影響を受けるので注意が必要である．

2.2.3 スタック・ポインタ(SP)

32ビット長のアドレス・レジスタA7がスタック・ポインタとして使われる．スタック・ポインタには，ユーザ・プログラムが実行される場合のユーザ・プログラム用と，スーパバイザ・プログラムが実行される場合のスーパバイザ用の2種類がある．68000がユーザ・プログラムを実行中のときにA7はユーザ・スタック・ポインタ(USP)となり，スーパバイザ・プログラムを実行中のときにA7はスーパバイザ・スタック・ポインタ(SSP)となる．

スタック・ポインタ(アドレス・レジスタA7)に対する操作は，他のアドレス・レジスタA0～A6に対する操作と同じで，スタック・ポインタをインデックス・レジスタとして使用することもできる．

2.2.4　プログラム・カウンタ（PC）

プログラム・カウンタは32ビット長で，そのうち24ビットがアドレスとして使われている．したがって，68000で生成できるアドレスは$000000から$FFFFFF(16,777,215)までで，直接16Mバイトのメモリのアクセスが可能である．

2.2.5　ステータス・レジスタ（SR）

図2.3に，ステータス・レジスタの構成を示す．ステータス・レジスタは16ビット長で，システム・バイト（上位8ビット）とユーザ・バイト（下位8ビット）に分かれている．各ビットの意味は次の通りである．

(1)　ビット0，キャリ(C)：加算の結果オペランドの最上位ビットからキャリが生じた場合，および減算の結果ボローが生じた場合に"1"がセットされ，それ以外はクリアされる．

(2)　ビット1，オーバフロー(V)：算術演算の結果が2の補数で表現できる範囲をオーバーした場合に"1"がセットされる．

(3)　ビット2，ゼロ(Z)：結果がゼロになった場合に"1"がセットされ，それ以外はクリアされる．

(4)　ビット3，ネガティブ(N)：結果の最上位ビットが"1"の場合に"1"がセットされ，それ以外はクリアされる．

(5)　ビット4，エクステンド(X)：倍精度演算におけるキャリとして働く．セット条件はCビットと同じである．

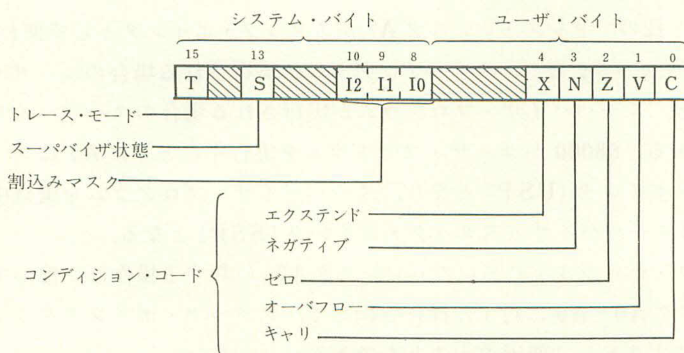


図 2.3　ステータス・レジスタの構成

(6) ビット8～10，割込みマスク(I0,I1,I2)：割込み制御用のマスク・ビットで，この3ビットで示される優先レベルよりも低いレベルの割込み要求はマスクされる．

(7) ビット13，スーパバイザ状態(S)：68000の処理状態がスーパバイザ状態にある(S=1)のか，ユーザ状態にある(S=0)のかを示す．

(8) ビット15，トレース・モード(T)：Tビットが"1"にセットされている場合，命令を実行するたびにスーパバイザ状態を示すSビットが"1"になり，エクセプション・ベクタ†としてベクタ・テーブルに与えたアドレスをもっているユーザの作ったトレース・ルーチンへジャンプする．トレース・ルーチンにより，メモリの内容やレジスタの内容を表示させたり，ステータス・レジスタの内容を調べるなどの種々のデバッグ処理を行なうことができる．

ステータス・レジスタの未使用ビットはすべて"0"となっている．

2.3 命令およびデータの構成とアドレシング

2.3.1 命令のフォーマット

図2.4に，68000の一般的な命令のフォーマットを示す．図に示すように，命令にはすべてオペレーション・ワードがあり，この1ワードのオペレーショ

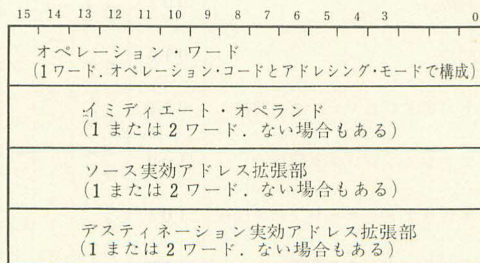


図 2.4 命令のフォーマット

ン・ワードで命令のタイプやアドレシング・モードが決定される．命令の長さは1～5ワード(2～10バイト)で，主にソースおよびデスティネーション・オペランドのアドレシング・モードの組合せで決まる．イミディエート・データ

† 例外ベクタと呼ぶ場合もある．ベクタ・テーブルについては表2.17を見よ．

やアブソリュート・アドレス，あるいはディスプレースメント付きのアドレシング・モードの場合には，オペレーション・ワードの後にそれぞれ必要な長さ(1または2ワード)の“アドレス拡張部”が確保される．最長の5ワードの命令の場合，オペレーション・ワードが1ワード，ソース・オペランドのアドレス拡張部が2ワード，それにデスティネーション・オペランドのアドレス拡張部が2ワードで，合計5ワードの命令となる．

図2.5に，オペレーション・ワードの一般的なフォーマットを示す．オペレーション・ワードの下位6ビットの実効アドレス・フィールドにオペランドの

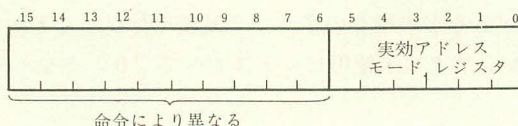


図 2.5　オペレーション・ワードのフォーマット

表 2.1　実効アドレス・フィールドのコード

アドレシング・モード	モード	レジスタ	備　　考
データ・レジスタ直接	000	レジスタ番号	レジスタ番号はデータ・レジスタの番号
アドレス・レジスタ直接	001	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
アドレス・レジスタ間接	010	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
ポストインクリメント・アドレス・レジスタ間接	011	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
プリデクリメント・アドレス・レジスタ間接	100	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
ディスプレースメント付きアドレス・レジスタ間接	101	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
インデックス付きアドレス・レジスタ間接	110	レジスタ番号	レジスタ番号はアドレス・レジスタの番号
アブソリュート・ショート	111	000	
アブソリュート・ロング	111	001	
ディスプレースメント付きプログラム・カウンタ相対	111	010	
インデックス付きプログラム・カウンタ相対	111	011	
イミディエート(またはステータス・レジスタ)	111	100	

アドレシング・モードがセットされる．6ビットの実効アドレス・フィールドは，モード・フィールド(3ビット)とレジスタ・フィールド(3ビット)の二つの領域に分かれており，各アドレシング・モードに応じて表2.1に示すビット・パターンがセットされる．

2.3.2 オペランドのサイズ

オペランドのサイズには，バイト(8ビット)，ワード(16ビット)，ロング・ワード(32ビット)がある．命令によりオペランドのサイズが固定的に決まっている場合を除けば，オペランドのサイズは命令の中の該当するビットで明確に規定される．

2.3.3 メモリ内のデータ構成

68000のメモリ内のワード構成を図2.6に示す．メモリにあるワード(命令も含む)およびロング・ワードのデータは常に偶数アドレス——$000000，$000002，……，$FFFFFE——でアクセスしなければならない．バイト・データの場合には，偶数，奇数いずれのアドレスでもバイト単位に個別にア

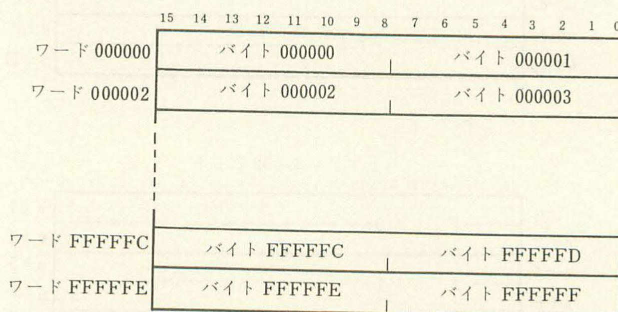
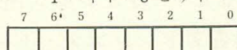


図 2.6 メモリ内のワード構成

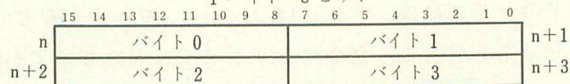
クセスすることができる．ワード(命令も含む)およびロング・ワードのデータも奇数アドレスでアクセスした場合は，アドレス・エラーの割込みが発生する．

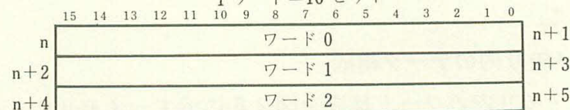
68000のサポートするデータは，ビット・データ，8・16・32ビットの整数データ，32ビットのアドレス・データおよびBCDデータである．これらのデータは，それぞれ図2.7に示すような形でメモリに格納される．

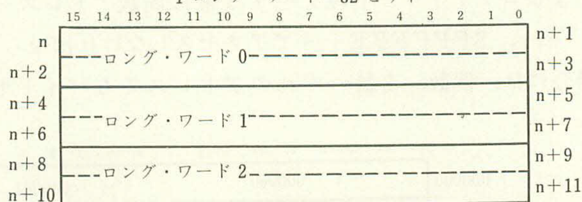
ビット・データ



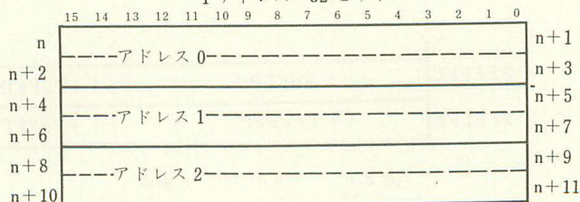
整数データ







アドレス・データ



BCDデータ

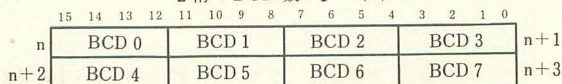


図 2.7　メモリ内のデータ構成

2.3.4 アドレシング・モード

68000のアドレシング・モードを表2.2に示す．表に示すように，68000は6形式14種類のアドレシング・モードを備えており，16ビット・マイクロコンピュータの中でも最も柔軟なアドレシング能力を備えているといえる．

(1) レジスタ直接アドレシング (register direct addressing)

このアドレシング・モードでは，16個あるデータ・レジスタもしくはアドレス・レジスタのうちの一つがオペランドとして指定される．

表 2.2 アドレシング・モード

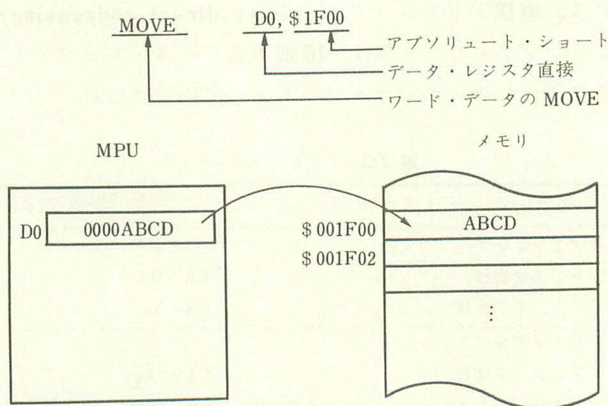
アドレシング・モード	実効アドレス
レジスタ直接アドレシング	
データ・レジスタ直接	EA＝Dn
アドレス・レジスタ直接	EA＝An
アドレス・レジスタ間接アドレシング	
アドレス・レジスタ間接	EA＝(An)
ポストインクリメント・アドレス・レジスタ間接	EA＝(An)，An←An＋N
プリデクリメント・アドレス・レジスタ間接	An←An－N，EA＝(An)
ディスプレースメント付きアドレス・レジスタ間接	EA＝(An)＋d16
インデックス付きアドレス・レジスタ間接	EA＝(An)＋(Xn)＋d8
アブソリュート・アドレシング	
アブソリュート・ショート	EA＝(次のワード)
アブソリュート・ロング	EA＝(次のロング・ワード)
プログラム・カウンタ相対アドレシング	
ディスプレースメント付きプログラム・カウンタ相対	EA＝(PC)＋d16
インデックス付きプログラム・カウンタ相対	EA＝(PC)＋(Xn)＋d8
イミディエート・データ・アドレシング	
イミディエート	データ：次のワードまたはロング・ワード
クイック・イミディエート	データ：命令の特定部分
インプライド・アドレシング	
インプライド・レジスタ	EA＝SR，USP，SSP，PC

EA：実効アドレス
Dn：データ・レジスタ
An：アドレス・レジスタ
Xn：インデックス・レジスタとして使用するデータ・レジスタまたはアドレス・レジスタ
PC：プログラム・カウンタ
SR：ステータス・レジスタ
USP：ユーザ・スタック・ポインタ
SSP：システム・スタック・ポインタ
d8：8ビット・ディスプレースメント
d16：16ビット・ディスプレースメント
N：バイトの場合は1，ワードの場合は2，ロング・ワードの場合は4
(　)：内容を表わす
←：置換え

i）　データ・レジスタ直接（data register direct）

オペランドは指定されたデータ・レジスタ自身で，，実行の対象となるデータはそのデータ・レジスタに格納されている．

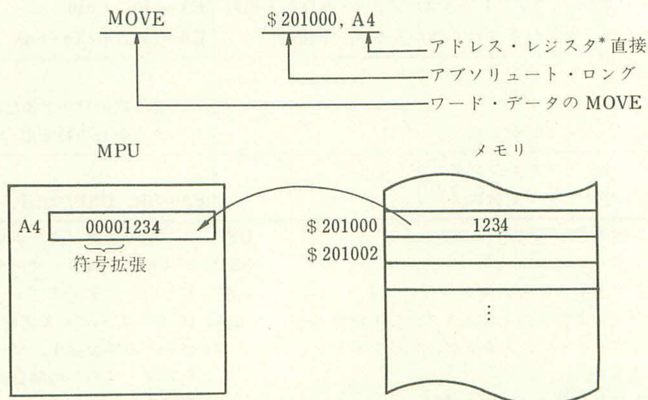
【例1】



ii）　アドレス・レジスタ直接（address register direct）

オペランドは指定されたアドレス・レジスタで，実行の対象となるデータはそのアドレス・レジスタに格納されている．

【例2】



＊アドレス・レジスタはロング・ワードとして符号拡張される

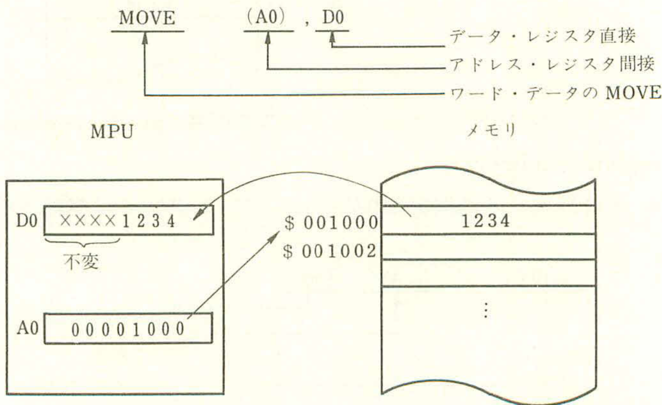
(2) アドレス・レジスタ間接アドレシング (address register indirect addressing)

指定されたアドレス・レジスタの内容がオペランドの実効アドレスを計算する場合のベース・アドレスとして使用される. アドレス・レジスタの内容と実効アドレスの関係は, アドレス・レジスタ間接アドレシングに属する五つのモードのどれが使われるかによって決定される.

i) アドレス・レジスタ間接 (address register indirect)

オペランドのアドレスは指定されたアドレス・レジスタに格納されており, アドレス・レジスタが示しているメモリ上のデータが命令の実行の対象となる.

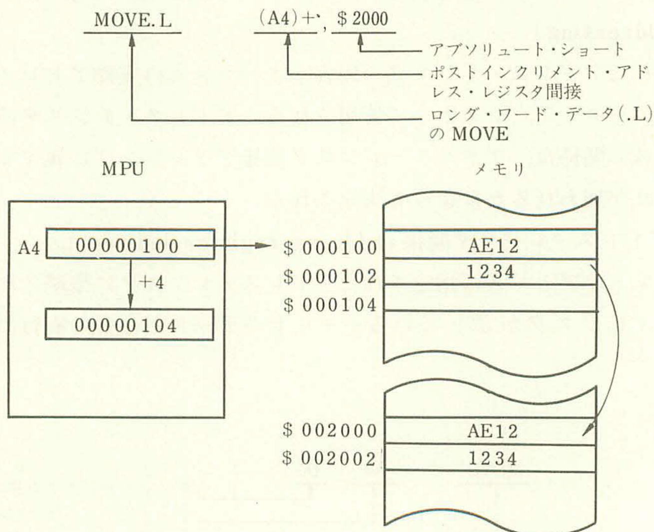
【例3】



ii) ポストインクリメント・アドレス・レジスタ間接 (postincrement address register indirect)

オペランドのアドレスは指定されたアドレス・レジスタに格納されており, アドレス・レジスタが示すメモリ上のデータが命令実行の対象となる. 命令実行後, バイトまたはワード, ロング・ワードのオペランドのサイズに応じて1または2, 4がインクリメントされる. ただし, アドレス・レジスタがスタック・ポインタでオペランドのサイズがバイトの場合には, ワード・バウンダリ (word boundary) を保持するために1ではなく2がインクリメントされる.

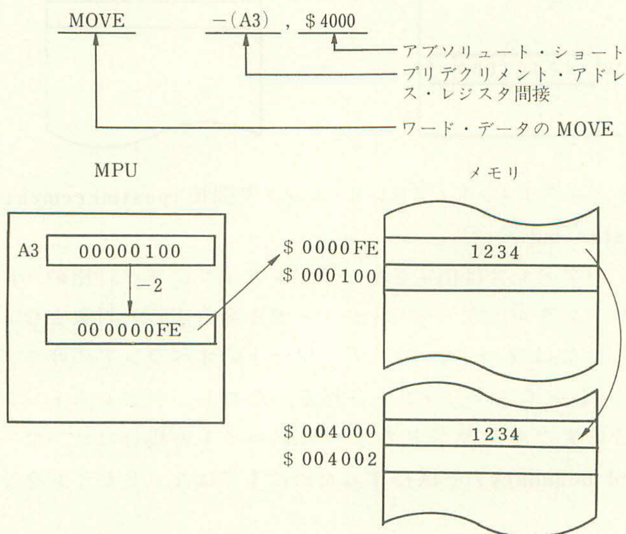
【例4】



iii)　プリデクリメント・アドレス・レジスタ間接(predecrement address register indirect)

オペランドのアドレスは指定されたアドレス・レジスタに格納されており，

【例5】

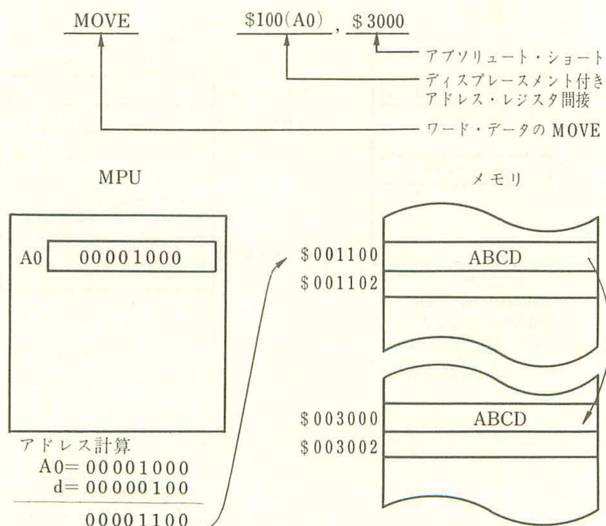


命令が実行される前に，バイト，ワード，またはロング・ワードのオペランドのサイズに応じて1または2，4がデクリメントされる．デクリメントされたアドレス・レジスタの内容が示すメモリのデータが命令実行の対象となる．ただし，アドレス・レジスタがスタック・ポインタでオペランドのサイズがバイトの場合には，スタック・ポインタのワード・バウンダリを保持するため1ではなく2がデクリメントされる．

iv) ディスプレースメント付きアドレス・レジスタ間接(address register indirect with displacement)

オペランドのアドレスは，指定されたアドレス・レジスタの内容とディスプレースメントの和で表わされ，和で示されるメモリのデータが命令実行の対象となる．ディスプレースメントは16ビットの符号付き整数(−32,768～+32,767)で，オペランドのアドレスを計算する場合には32ビットに符号拡張される．

【例6】



v) インデックス付きアドレス・レジスタ間接(address register indirect with index)

オペランドのアドレスは，指定されたアドレス・レジスタ，インデックス・レジスタ，およびディスプレースメントの総和で示され，その総和が示すメモリのデータが命令実行の対象となる．

インデックス・レジスタには，データ・レジスタまたはアドレス・レジスタを指定する．インデックス・レジスタのサイズはワードまたはロング・ワードで，下位16ビットのみを使用するのか，32ビットすべてを使用するのかを選択することができる．下位16ビットを使用する場合には32ビットに符号拡張されてアドレスの計算が行なわれる．

ディスプレースメントは8ビットの符号付き整数（−128～+127）で，アドレスを計算する場合には32ビットに符号拡張される．

【例7】

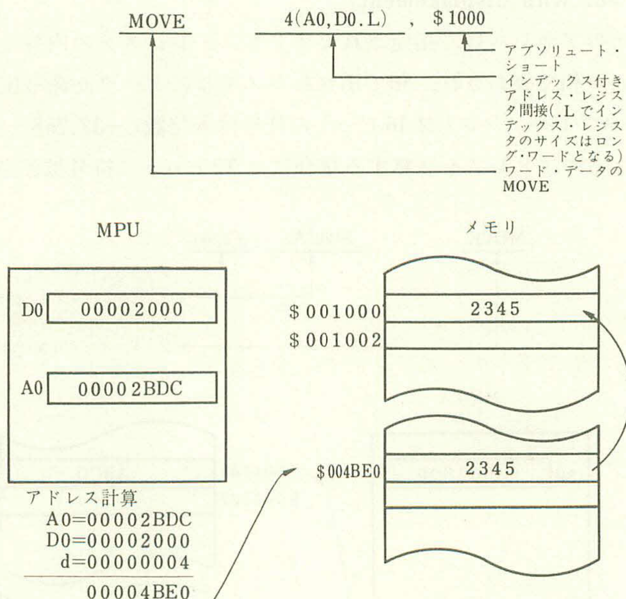


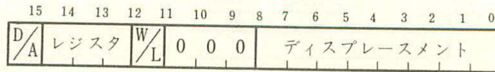
図2.8に，このアドレシング・モードのアドレス拡張部のフォーマットを示す．

(3)　アブソリュート・アドレシング（absolute addressing）

このアドレシング・モードには，ショートおよびロングの二つのアドレシング・モードがあり，前者では命令に含まれるアドレスが16ビット（32ビットへ符号拡張される），後者では32ビットのアドレス全体が命令に含まれる．

i)　アブソリュート・ショート（absolute short）

指定した絶対アドレスが示すメモリのデータが命令実行の対象となる．この



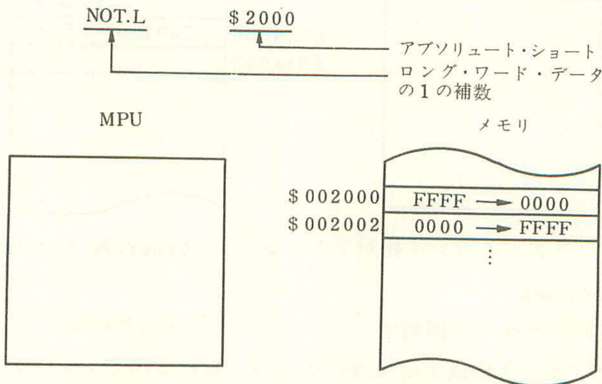
D/A　　：インデックス・レジスタがデータ・レジスタの場合0，インデックス・レジスタがアドレス・レジスタの場合1
レジスタ：データ・レジスタ，アドレス・レジスタの番号
W/L　　：インデックス・レジスタのサイズがワードの場合0，インデックス・レジスタのサイズがロング・ワードの場合1

図 2.8 アドレス拡張部のフォーマット
（インデックス付きアドレス・レジスタ間接，インデックス付きプログラム・カウンタ相対の場合）

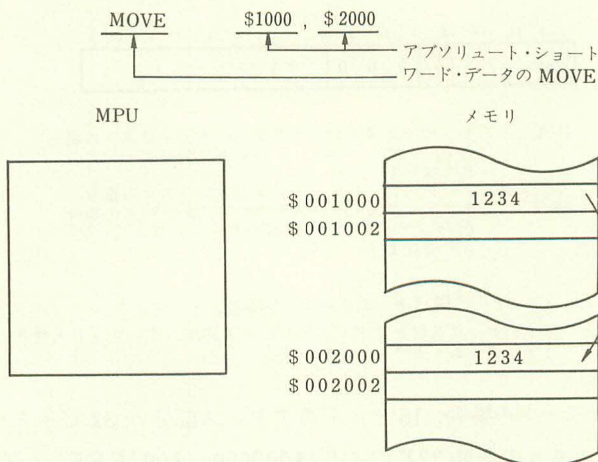
アドレシング・モードの場合，16ビットのアドレス部分が32ビットへ符号拡張されるため，メモリの先頭32Kバイト（$000000～$007FFF）と最後の32Kバイト（$FF8000～$FFFFFF）がアクセス可能なメモリ領域となる．

アブソリュート・ショート・アドレシングを使用した場合には，アブソリュート・ロング・アドレシングの場合に比べ命令が1ワード短くなり，命令の実行時間も4サイクルだけ短くなる．このことは，メモリの両端の32Kバイトに頻繁に使われるデータやテンポラリ・データを置き，ショート・アドレシングを使用してメモリ・アクセスを行なうようにすれば，より効率的なプログラムの作成が可能になることを意味している．

【例8】



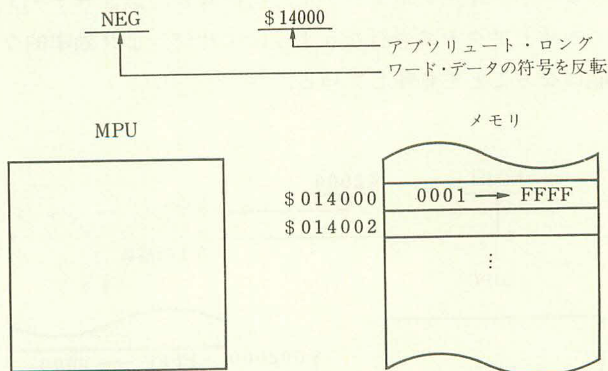
【例9】



ii）アブソリュート・ロング（absolute long）

指定した絶対アドレスが示すメモリのデータが命令実行の対象となる．この場合，16Mバイトのアドレス空間の任意のアドレスのアクセスが可能で，もちろんメモリの先頭32Kバイトと最後の32Kバイトのアクセスも可能である．

【例10】



(4) プログラム・カウンタ相対アドレシング（program counter relative addressing）

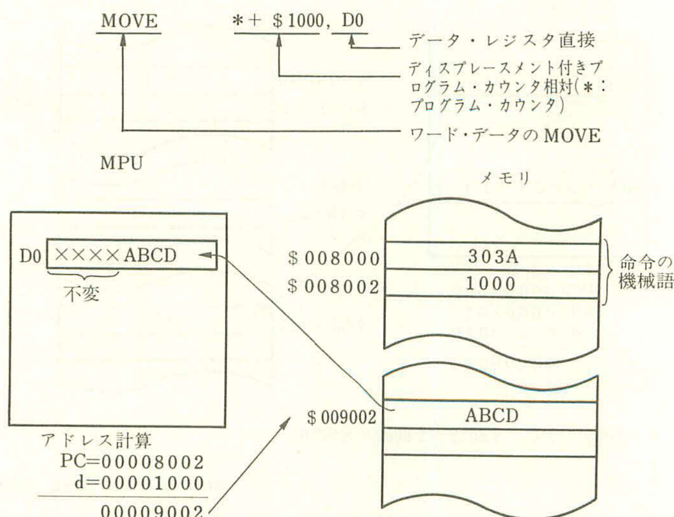
プログラム・カウンタ相対アドレシング・モードはリロケータブル（relocatable）なプログラムを作成する場合によく用いられるアドレシング・モードで，68000には2種類のプログラム・カウンタ相対アドレシング・モードがある．

このアドレシング・モードを，命令の実行により内容が更新されるようなデータのアドレシング・モード (MOVE, ADD 命令のデスティネーション・オペランドやCLR命令のオペランド)として指定することはできない．

i) ディスプレースメント付きプログラム・カウンタ相対 (program counter relative with displacement)

オペランドのアドレスは，プログラム・カウンタとディスプレースメントの和で表わされる．ディスプレースメントは16ビットの符号付き整数で，数値の範囲は−32768～+32767となる．したがって，実行中の命令のあるアドレスから上に32Kバイト，下に32Kバイトが指定可能なアドレスの範囲となる．

【例11】



この命令の実行時にはプログラム・カウンタの内容が$008002となる．

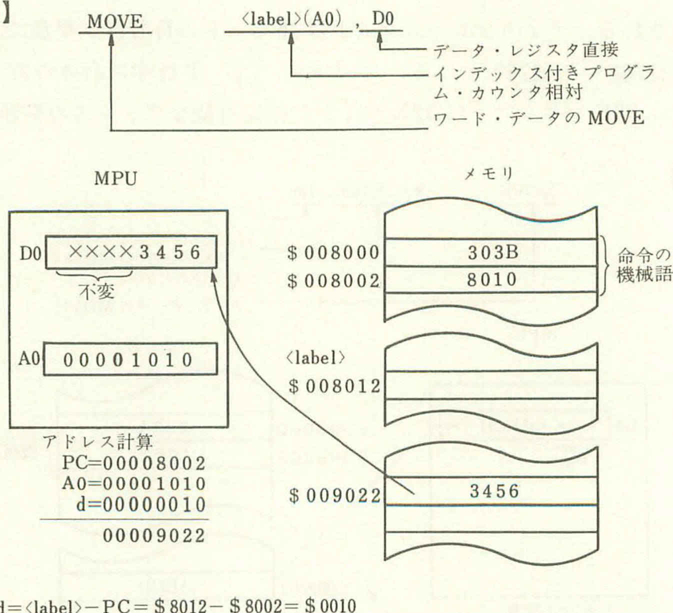
ii) インデックス付きプログラム・カウンタ相対 (program counter relative with index)

オペランドのアドレスは，プログラム・カウンタとインデックス・レジスタとディスプレースメントの和で表わされる．インデックス・レジスタとしてはデータ・レジスタかアドレス・レジスタのいずれかを指定する．インデックス・レジスタのサイズはワードまたはロング・ワードで，下位16ビットを使用するのか，32ビットすべてを使用するのかは選択することができる．下位16ビッ

トが指定された場合は，32ビットに符号拡張されてアドレス計算が行なわれる．ディスプレースメントは8ビットの符号付き整数（−128〜+127）で，アドレス計算の場合にはやはり32ビットに符号拡張される．

このアドレシング・モードのアドレス拡張部のフォーマットは，インデックス付きアドレス・レジスタ間接の場合と同じになる（図2.8）．

【例12】



(5)　イミディエート・データ・アドレシング（immediate data addressing）

このアドレシング・モードではオペランドとして扱うデータを命令で直接指定する．

i)　イミディエート（immediate）

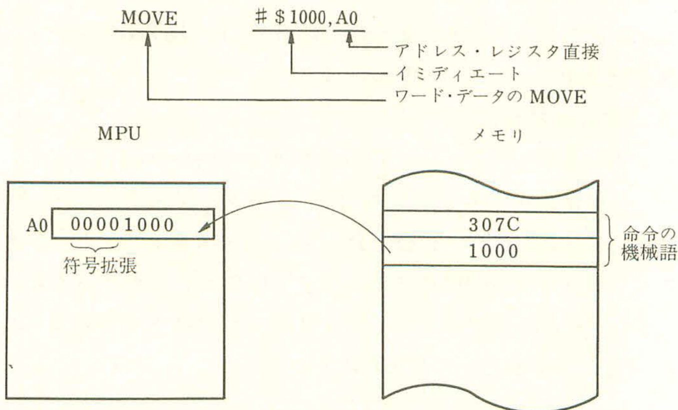
イミディエート・データとしては，バイト，ワード，ロング・ワードの各データが指定可能である．

ii)　クイック・イミディエート（quick immediate）

このアドレシング・モードでは，命令のオペレーション・ワードの中にイミディエート・データが組み込まれている．したがって，オペレーション・ワードの後に続くイミディエート・オペランドの部分がなくなり，命令の長さがイミ

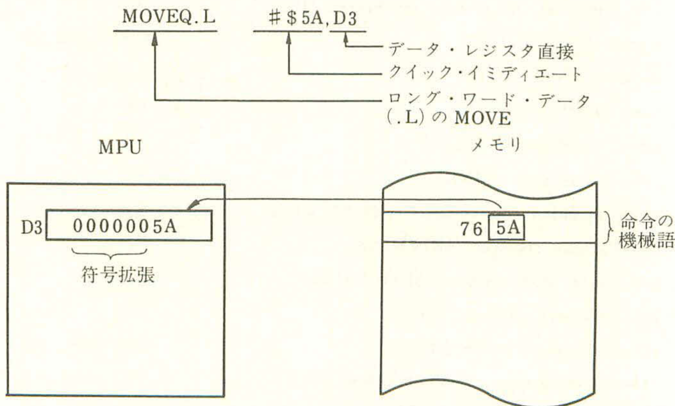
ディエート・アドレシングの場合に比べ1ないし2ワード短くなる.

【例13】



このアドレシング・モードの使える命令は, ADDQ, SUBQ, MOVEQ の三つであるが, イミディエート・データの大きさなどに制限があるので注意が必要である.

【例14】



iii) コンディション・コードとステータス・レジスタ

イミディエート・データ・アドレシング・モードの特別の場合として, 次のような場合がある.

ANDI, ORI, EORI の三つの命令で, オペレーション・ワードの実効アドレ

ス・フィールドでイミディエート・データ・アドレシング・モードが指定された場合は，デスティネーション・オペランドとしてステータス・レジスタが参照される．命令で実行するデータ・サイズがバイトの場合にはコンディション・コードが参照され，ワードの場合にはステータス・レジスタ全体が参照される(ワードの場合は特権命令)．

(6)　インプライド・アドレシング (implied addressing)

命令には，プログラム・カウンタ(PC)，スタック・ポインタ(SP)，スーパバイザ・スタック・ポインタ(SSP)，ユーザ・スタック・ポインタ(USP)，ステータス・レジスタ(SR)の各レジスタを暗黙的に参照するものがある．表2.3に，インプライド・アドレシングを使用する命令とその対象となるレジスタを示す(SRの下位8ビットをCCRという)．

表 2.3　インプライド・アドレシングの命令とレジスタ

命　　令	インプライド・レジスタ
branch conditional (B_{CC}), branch always (BRA)	PC
branch to subroutine (BSR)	PC, SP
check register against bounds (CHK)	SSP, SR
test condition, decrement and branch (DB_{CC})	PC
signed divide (DIVS)	SSP, SR
unsigned divide (DIVU)	SSP, SR
jump (JMP)	PC
jump to subroutine (JSR)	PC, SP
link and allocate (LINK)	SP
move condition code register (MOVE CCR)	SR
move status register (MOVE SR)	SR
move user stack pointer (MOVE USP)	USP
push effective address (PEA)	SP
return from exception (RTE)	PC, SP, SR
return and restore condition codes (RTR)	PC, SP, SR
return from subroutine (RTS)	PC, SP
trap (TRAP)	SSP, SR
trap on overflow (TRAPV)	SSP, SR
unlink (UNLK)	SP

2.4 命令セット

68000には，変化型を含めて77の命令（表2.4）がある．この命令を機能別

表 2.4 68000の命令

命令	内容	命令	内容
ABCD	add decimal with extend	EXT	sign extend
ADD	add binary	JMP	jump
ADDA*1	add address	JSR	jump to subroutine
ADDI*1	add immediate	LEA	load effective address
ADDQ*1	add quick	LINK	link and allocate
ADDX*1	add extend	LSL	logical shift left
AND	AND logical	LSR	logical shift right
ANDI*2	AND immediate	MOVE	move data from source to destination
ASL	arithmetic shift left	MOVE to CCR*5	move to condition code register
ASR	arithmetic shift right	MOVE to SR*5	move to the status register
B_{CC}	branch conditionally	MOVE from SR*5	move from the status register
BCHG	test a bit and change	MOVE USP*5	move user stack pointer
BCLR	test a bit and clear	MOVEA*5	move address
BRA	branch always	MOVEM	move multiple registers
BEST	test a bit and set	MOVEP	move peripheral data
BSR	branch to subroutine	MOVEQ*5	move quick
BTST	test a bit	MULS	signed multiply
CHK	check register against bounds	MULU	unsigned multiply
CLR	clear on operand	NBCD	negate decimal with extend
CMP	compare	NEG	negate
CMPA*3	compare address	NEGX*6	negate with extend
CMPI*3	compare immediate	NOP	no operation
CMPM*3	compare memory	NOT	logical complement
DB_{CC}	test condition, decrement and branch	OR	inclusive OR logical
DIVS	signed divide	ORI*7	inclusive OR immediate
DIVU	unsigned divide	PEA	push effective address
EOR	exclusive OR logical	RESET	reset external devices
EORI*4	exclusive OR immediate	ROL	rotate left (without extend)
EXG	exchange registers	ROR	rotate right (without extend)

（次ページに続く）

（表2.4の続き）

命　令	内　　容	命　令	内　　容
ROXL	rotate left with extend	SUBI*8	subtract immediate
ROXR	rotate right with extend	SUBQ*8	subtract quick
RTE	return from exception	SUBX*8	subtract with extend
RTR	return and restore condition codes	SWAP	swap register halves
RTS	return from subroutine	TAS	test and set on operand
SBCD	subtract decimal with extend	TRAP	trap
Scc	set according to condition	TRAPV	trap on overflow
STOP	load status register and stop	TST	test on operand
SUB	subtract binary	UNLK	unlink
SUBA*8	subtract address		

＊1：ADDの変化型　＊5：MOVEの変化型
＊2：ANDの変化型　＊6：NEGの変化型
＊3：CMPの変化型　＊7：ORの変化型
＊4：EORの変化型　＊8：SUBの変化型

に分類すると次のようになる．

(1)　データ転送命令
(2)　整数算術演算命令
(3)　論理演算命令
(4)　シフトおよびローテート命令
(5)　ビット処理命令
(6)　2進化10進数（BCD）処理命令
(7)　プログラム制御命令
(8)　システム制御命令

以下では，68000の命令を機能別分類に従って説明する．以下の表2.5～表2.13に使われている符号の意味は，次の通りである．

An：アドレス・レジスタ
Dn：データ・レジスタ
Rn：アドレス・レジスタまたはデータ・レジスタ
SP：スタック・ポインタ
X　：エクステンド（コンディション・コード）
Z　：ゼロ（コンディション・コード）
C　：キャリ（コンディション・コード）
#×××：イミディエート・データ

disp：ディスプレースメント
s　：ソース
d　：デスティネーション
EA：実効アドレス
[ ]：ビット番号
( )：内容
＋：加算
－：減算

＊：乗算
／：除算
∧：論理積（AND）
∨：論理和（OR）
⊕：排他的論理和（EOR）
～：反転
ⓐ＋：ポストインクリメント
ⓐ－：プリデクリメント
⟷：交換
⟶：置換え

2.4.1 データ転送命令 (data movement operation)

データ転送命令の中で最も基本的な命令はMOVE命令である．MOVE命令を使えば，あらゆるデータのあらゆる転送が可能である．すなわち，バイト，ワード，ロング・ワード長のデータを，メモリからメモリへ，メモリからレジスタへ，レジスタからメモリへ，そしてレジスタからレジスタへ転送することができる．

このほか，データ転送の命令としてはMOVEM(move multiple registers)，MOVEP (move peripheral data)，EXG(exchange registers)，LEA(load effective address)，PEA (push effective address)，LINK (link and allocate)，UNLK (unlink)， MOVEQ (move quick) がある．

表 2.5 データ転送命令

命　令	オペランド・サイズ	オペレーション	備　　考
EXG	32	Rx⟷Ry	レジスタ間のデータの交換
LEA	32	EA⟶An	実効アドレスをアドレス・レジスタへロード
LINK	－	An⟶SP @－ SP⟶An SP＋disp⟶SP	現在のスタック・ポインタをアドレス・レジスタへ退避して，必要な大きさのワーク・エリアをスタックへ確保
MOVE	8，16，32	$(EA)_s$⟶EAd	データの転送
MOVEA	16，32	(EA) ⟶An	アドレス・データの転送
MOVEM	16，32	(EA) ⟶An，Dn An，Dn⟶EA	データ・レジスタ，アドレス・レジスタの退避／回復
MOVEP	16，32	(EA) ⟶Dn Dn⟶EA	データの入出力
MOVEQ	32	＃×××⟶Dn	イミディエート・データ(－128～＋127)のロード
PEA	32	EA⟶SP @－	実効アドレスをスタックへプッシュ
SWAP	32	Dn[31：16]⟷Dn[15：0]	上位ワードと下位ワードの入換え
UNLK	－	An⟶SP SP @＋⟶An	スタックへ確保したワーク・エリアの解放(LINKの逆の命令)

2.4.2 整数算術演算命令 (integer arithmetic operation)

加算，減算，乗算，除算の基本的な算術演算命令のほか，表2.6に示すような命令がある．

表 2.6　整数算術演算命令

命　令	オペランド・サイズ	オペレーション	備　　考
ADD	8，16，32	Dn+(EA)⟶Dn (EA)+Dn⟶EA	データ・レジスタの加算
ADDA	16，32	An+(EA)⟶An	アドレス・レジスタの加算
ADDI	8，16，32	(EA)+#×××⟶EA	イミディエート・データの加算
ADDQ	8，16，32	(EA)+#×××⟶EA	イミディエート・データ (1～8) の加算
ADDX	8，16，32	Dx+Dy+X⟶Dx Ax@−+Ay@−+X⟶Ax	Xフラグを伴う加算
CLR	8，16，32	0⟶EA	オペランドのゼロ・クリア
CMP	8，16，32	Dn−(EA)	データ・レジスタのデータの比較
CMPA	16，32	An−(EA)	アドレス・レジスタの比較
CMPI	8，16，32	(EA)−#×××	イミディエート・データによる比較
CMPM	8，16，32	Ax@+−Ay@+	メモリ間のデータの比較
DIVS	32÷16	Dn/(EA)⟶Dn	符号付き除算
DIVU	32÷16	Dn/(EA)⟶Dn	符号なし除算
EXT	8→16 16→32	$(Dn)_8$ ⟶ Dn_{16} $(Dn)_{16}$ ⟶ Dn_{32}	データ・レジスタ内のデータの拡張
MULS	16*16→32	Dn*(EA)⟶Dn	符号付き乗算
MULU	16*16→32	Dn*(EA)⟶Dn	符号なし乗算
NEG	8，16，32	0−(EA)⟶EA	データの符号の反転
NEGX	8，16，32	0−(EA)−X⟶EA	Xフラグを伴うデータの符号の反転
SUB	8，16，32	Dn−(EA)⟶Dn (EA)−Dn⟶EA	データ・レジスタの減算
SUBA	16，32	An−(EA)⟶An	アドレス・レジスタの減算
SUBI	8，16，32	(EA)−#×××⟶EA	イミディエート・データの減算
SUBQ	8，16，32	(EA)−#×××⟶EA	イミディエート・データ (1～8) の減算
SUBX	8，16，32	Dx−Dy−X⟶Dx Ax@−−Ay@−−X⟶Ax	Xフラグを伴う減算
TAS	8	(EA)−0，1⟶EA[7]	テスト・アンド・セット
TST	8，16，32	(EA)−0	オペランドのテスト

乗算と除算の場合には，それぞれ符号付きと符号なしで演算を行なう命令が準備されている．また，倍精度演算やビット長の異なるデータの演算のためには，ADDX，SUBX や EXT，NEGX の命令が準備されている．

このほか，マルチプロセッサ・システムを実現する上で不可欠な TAS (test and set) 命令がある．

2.4.3 論理演算命令 (logical operation)

表 2.7 に論理演算命令を示す．

表 2.7 論理演算命令

命　令	オペランド・サイズ	オペレーション	備　　考
AND	8，16，32	Dn∧(EA)⟶Dn (EA)∧Dn⟶EA	データ・レジスタ(と)の論理積
ANDI	8，16，32	(EA)∧#×××⟶EA	イミディエート・データとの論理積
EOR	8，16，32	Dn⊕(EA)⟶Dn (EA)⊕Dn⟶EA	データ・レジスタ(と)の排他的論理和
EORI	8，16，32	(EA)⊕#×××⟶EA	イミディエート・データとの排他的論理和
OR	8，16，32	Dn∨(EA)⟶Dn (EA)∨Dn⟶EA	データ・レジスタ(と)の論理和
ORI	8，16，32	(EA)∨#×××⟶EA	イミディエート・データとの論理和
NOT	8，16，32	~(EA)⟶EA	1の補数を作成

2.4.4 シフトおよびローテート命令 (shift and rotate operation)

表 2.8 に命令を示す．全命令ともレジスタもしくはメモリの内容のシフト(ローテイト)が可能であるが，メモリの場合には，オペランドのサイズはワードでシフト(ローテイト)数も 1 ビットに限定される．レジスタの場合には，バイト，ワード，ロング・ワードの各データに対して，別のデータ・レジスタで指定するビット数(0～63 ビット)もしくは命令で指定するビット数(1～8 ビット)だけのシフト(ローテイト)が可能である．

2.4.5 ビット処理命令 (bit manipulation operation)

ビット処理命令としては，表 2.9 に示す 4 種類の命令が準備されている．メモリとデータ・レジスタの内容がビット処理の対象となるが，メモリの場合はオペランドのサイズはバイトに，データ・レジスタの場合はロング・ワード(すなわちレジスタ全体) に限られる．

表 2.8　シフトおよびローテート命令

命令	オペランド・サイズ	オペレーション	備考
ASL	8, 16, 32	X/C ← [←] ← 0	メモリのときはサイズはワードで1ビットのみシフト
ASR	8, 16, 32	[→] → X/C（最上位ビットを保持）	メモリのときはサイズはワードで1ビットのみシフト
LSL	8, 16, 32	X/C ← [←] ← 0	メモリのときはサイズはワードで1ビットのみシフト
LSR	8, 16, 32	0 → [→] → X/C	メモリのときはサイズはワードで1ビットのみシフト
ROL	8, 16, 32	C ← [←]（最上位ビットを最下位へ）	メモリのときはサイズはワードで1ビットのみローテート
ROR	8, 16, 32	[→] → C（最下位ビットを最上位へ）	メモリのときはサイズはワードで1ビットのみローテート
ROXL	8, 16, 32	C ← [←] ← X（最上位ビットをXへ）	メモリのときはサイズはワードで1ビットのみローテート
ROXR	8, 16, 32	X → [→] → C（最下位ビットをXへ）	メモリのときはサイズはワードで1ビットのみローテート

表 2.9　ビット処理命令

命令	オペランド・サイズ	オペレーション	備表
BTST	8, 32	~bit of (EA) ⟶ Z	ビットの 1, 0 をテスト
BSET	8, 32	~bit of (EA) ⟶ Z 1 ⟶ bit of (EA)	ビットに1をセット
BCLR	8, 32	~bit of (EA) ⟶ Z 0 ⟶ bit of (EA)	ビットをゼロ・クリア
BCHG	8, 32	~bit of (EA) ⟶ Z ~bit of (EA) ⟶ bit of (EA)	ビットの 1, 0 を反転

2.4.6　2進化10進数処理命令（binary coded decimal operation）

10進数の演算を行なうために2進化10進数とその算術演算が得られた．2進化10進数（BCD）の算術演算は，表2.10に示す命令を使って行なわれる．

表 2.10　2進化10進数（BCD）処理命令

命　令	オペランド・サイズ	オペレーション	備　　考
ABCD	8	$Dm_{10}+Dn_{10}+X \rightarrow Dm$ $Am@-_{10}+An@-_{10}+X \rightarrow Am$	BCDデータの加算
SBCD	8	$Dm_{10}-Dn_{10}-X \rightarrow Dm$ $Am@-_{10}+An@-_{10}-X \rightarrow An$	BCD データの減算
NBCD	8	$0-(EA)_{10}-X \rightarrow EA$	BCD データの補数の作成

2.4.7 プログラム制御命令（program control operation）

プログラム制御命令を表2.11に示す．

表 2.11　プログラム制御命令

命　令	オペレーション
条件付き	
B_{CC}	条件付きブランチ（14条件） 8ビットおよび16ビット・ディスプレースメント
CB_{CC}	条件テスト，カウンタ・デクリメント & ブランチ 16ビット・ディスプレースメント
S_{CC}	条件付きバイト・セット（16条件）
条件なし	
BRA	無条件ブランチ 8ビットおよび16ビット・ディスプレースメント
BSR	サブルーチンへブランチ 8ビットおよび16ビット・ディスプレースメント
JMP	ジャンプ
JSR	サブルーチンへジャンプ
リターン	
RTR	リターン & コンディション・コード回復
RTS	サブルーチンからリターン

条件付き命令（conditional instruction）でテストされるコンディションには表2.12に示すようなものがある．

表 2.12　条件付き命令のコンディション

ニモニック	コンディション	テ　ス　ト	ニモニック	コンディション	テ　ス　ト
T*	true	1	VC	overflow clear	$\overline{V}$
F*	false	0	VS	overflow set	V
HI	high	$\overline{C}\cdot\overline{Z}$	PL	plus	$\overline{N}$
LS	low or same	$C+Z$	MI	minus	N
CC	carry clear	$\overline{C}$	GE	greater or equal	$N\cdot V+\overline{N}\cdot\overline{V}$
CS	carry set	C	LT	less than	$N\cdot\overline{V}+\overline{N}\cdot V$
NE	not equal	$\overline{Z}$	GT	greater than	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$
EQ	equal	Z	LE	less or equal	$Z+N\cdot\overline{V}+\overline{N}\cdot V$

*　B_{CC} 命令では使用できない
C：キャリ　　N：ネガティブ
Z：ゼロ　　V：オーバフロー

表 2.13　システム制御命令

命　　令	オペレーション
特権命令	
RESET	外部デバイスのリセット
RTE	エクセプションからリターン
STOP	プログラム実行停止
ANDI to SR	ステータス・レジスタの AND
EORI to SR	ステータス・レジスタの EOR
ORI to SR	ステータス・レジスタの OR
MOVE USP	ユーザ・スタック・ポインタの転送
MOVE EA to SR	新しいステータスをセット
トラップ発生	
TRAP	トラップ
TRAPV	オーバフローの状態に応じてエクセプション発生
CHK	レジスタの境界値をチェック
ステータス・レジスタ	
ANDI to CCR	コンディション・コード・レジスタの AND
EORI to CCR	コンディション・コード・レジスタの EOR
ORI to CCR	コンディション・コード・レジスタの OR
MOVE EA to CCR	新しいコンディション・コードをセット
MOVE SR to EA	ステータス・レジスタを転送

SR：status register
USP：user stack pointer
CCR：condition code register

2.4.8 システム制御命令 (system control operation)

システムの制御は，特権命令(privileged instruction)，トラップ発生命令，およびステータス・レジスタを書き換える命令によってなされる．

表2.13にシステム制御命令を示した．

2.5 入出力信号

68000の入出力信号について説明する．68000の入出力信号は，図2.9に示すようなグループに機能的に分類することができる．表2.14に各信号をまとめて示す．以下，各信号について説明する．

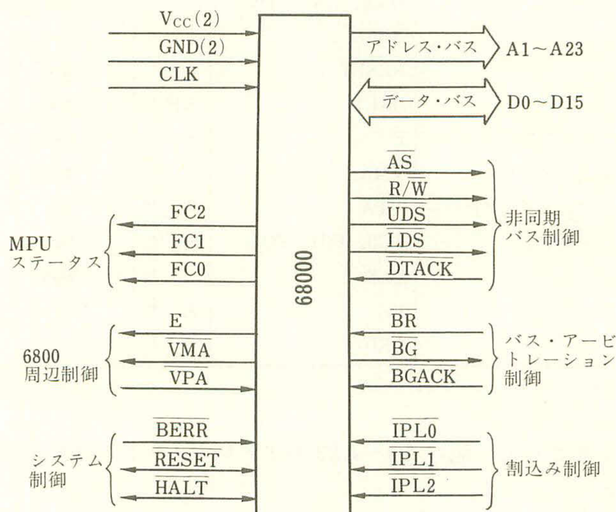


図 2.9 入出力信号

2.5.1 アドレス・バス (A1～A23)

アドレス・バスは，23本の出力用スリー・ステート・バスで，8Mワード(16Mバイト)のデータをアドレスすることができる．

アドレス・バスは，通常命令のフェッチやデータのリード/ライトを行なうためのバス・オペレーション用アドレスを出力する信号線として使われるが，割込みサイクルの場合には，アドレス・ライン A1, A2, A3 に受け付けた割込

図 2.14　入出力信号一覧表

信　号　名	記　号	入出力	アクティブ状態	スリー・ステート
アドレス・バス	A1～A23	出　力	high	yes
データ・バス	D0～D15	入出力	high	yes
アドレス・ストローブ	$\overline{\text{AS}}$	出　力	low	yes
リード/ライト	R/$\overline{\text{W}}$	出　力	リード時：high ライト時：low	yes
上位および下位データ・ストローブ	$\overline{\text{UDS}}$, $\overline{\text{LDS}}$	出　力	low	yes
データ転送アクノリッジ	$\overline{\text{DTACK}}$	入　力	low	no
バス・リクエスト	$\overline{\text{BR}}$	入　力	low	no
バス・グラント	$\overline{\text{BG}}$	出　力	low	no
バス・グラント・アクノリッジ	$\overline{\text{BGACK}}$	入　力	low	no
割込み優先レベル	$\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL2}}$	入　力	low	no
バス・エラー	$\overline{\text{BERR}}$	入　力	low	no
リセット	$\overline{\text{RESET}}$	入出力	low	no*
ホルト	$\overline{\text{HALT}}$	入出力	low	no*
イネーブル	E	出　力	high	no
バリッド・メモリ・アドレス	$\overline{\text{VMA}}$	出　力	low	yes
バリッド・ペリフェラル・アドレス	$\overline{\text{VPA}}$	入　力	low	no
ファンクション・コード	FC0, FC1, FC2	出　力	high	yes
クロック	CLK	入　力	high	no
電源入力	V_{CC}	入　力	—	—
グランド	GND	入　力	—	—

*　オープン・ドレイン

みのレベルが示される．他のA4～A23のアドレス・ラインはすべて"high"にセットされる．

2.5.2　データ・バス(D0～D15)

データ・バスは，16本の入出力用のスリー・ステート・バスで，ワードまたはバイト長のデータの転送を行なう汎用データ・バスとして使われる．

割込みアクノリッジ・サイクルの場合には，外部デバイスが送出した割込みベクタの番号がデータ・ラインD0～D7にセットされる．

2.5.3　非同期バス制御

MPUと周辺デバイスとの間の非同期データの転送は，以下の五つの制御信号を使って行なわれる．

(1) アドレス・ストローブ($\overline{\mathrm{AS}}$)

この信号は，アドレス・バスに有効なアドレスがあることを示す.

(2) リード/ライト(R/$\overline{\mathrm{W}}$)

この信号は，データ・バスの転送方向がリード・サイクル，ライト・サイクルのいずれのサイクルであるかを示す. R/$\overline{\mathrm{W}}$は，次の上位および下位データ・ストローブと一緒になって機能する.

(3) 上位および下位データ・ストローブ($\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$)

この二つの信号は，表2.15に示すようなデータ・バス上のデータの制御を行なう. R/$\overline{\mathrm{W}}$が“high”の場合，MPUはデータ・バスの読込みを行ない，R/$\overline{\mathrm{W}}$が“low”の場合，MPUはデータ・バスへ書込みを行なう.

表 2.15 データ・バスのデータ・ストローブ制御

$\overline{\mathrm{UDS}}$	$\overline{\mathrm{LDS}}$	R/$\overline{\mathrm{W}}$	D8～D15	D0～D7
high	high	—	無効データ	無効データ
low	low	high	データ・ビット 8～15	データ・ビット 0～7
high	low	high	無効データ	データ・ビット 0～7
low	high	high	データ・ビット 8～15	無効データ
low	low	low	データ・ビット 8～15	データ・ビット 0～7
high	low	low	データ・ビット 0～7*	データ・ビット 0～7
low	high	low	データ・ビット 8～15	データ・ビット 8～15*

* この状態は現在の実行結果であり，将来のデバイスでは発生しない場合もある

(4) データ転送アクノリッジ($\overline{\mathrm{DTACK}}$)

この入力信号は，データの転送が完了したことを示す. MPUがリード・サイクル中に$\overline{\mathrm{DTACK}}$を認識するとデータ・バス上のデータをラッチし，バス・サイクルを終了させる. ライト・サイクル中に$\overline{\mathrm{DTACK}}$を認識するとバス・サイクルを終了させる.

2.5.4 バス・アービトレーション制御

次の三つの信号によって，バス・マスタ・デバイスを決定するバス・アービトレーション制御を行なう.

(1) バス・リクエスト($\overline{\mathrm{BR}}$)

この入力信号には，バス・マスタとなりうるすべてのデバイスがワイヤード・オア(wired OR)接続される. この入力によってデバイスがMPUにバス・マ

スタとなることを要求していることを示す.

(2)　バス・グラント($\overline{\mathrm{BG}}$)

この出力信号は，MPUが現在のバス・サイクルの終了した時点でバスの制御を解放し，バス・リクエストによりバス・マスタとなることを要求しているデバイスにバス・マスタを移すことを示す.

(3)　バス・グラント・アクノリッジ($\overline{\mathrm{BGACK}}$)

この入力信号は，バス・マスタとなることを要求していたデバイスが，バス・マスタとなったことを示す．この信号は，次の四つの条件が満たされるまでアクティブ状態にはならない.

i)　バス・グラント($\overline{\mathrm{BG}}$)を受信している.

ii)　アドレス・ストローブ($\overline{\mathrm{AS}}$)がインアクティブになり，したがってMPUがバスを使用していない.

iii)　データ転送アクノリッジ($\overline{\mathrm{DTACK}}$)がインアクティブになり，したがってメモリあるいは周辺デバイスがバスを使用していない.

iv)　バス・グラント・アクノリッジがインアクティブになり，したがってバス・マスタとなったデバイスが現在存在しない.

2.5.5　割込み制御($\overline{\mathrm{IPL0}}$，$\overline{\mathrm{IPL1}}$，$\overline{\mathrm{IPL2}}$)

この三本の入力信号は，割込みを要求しているデバイスの優先レベルを示すのに使われる．レベル7が優先度が最も高く，レベル0は割込み要求がないことを示す．$\overline{\mathrm{IPL0}}$が最下位ビットで$\overline{\mathrm{IPL2}}$が最上位ビットとなる.

2.5.6　システム制御

MPUをリセットあるいはホールトする場合，また，バス・エラーの発生をMPUに知らせる場合にシステム制御入力を使用する.

(1)　バス・エラー($\overline{\mathrm{BERR}}$)

この入力信号は，現在実行中のバス・サイクルで問題が発生したことをMPUに知らせる信号である.

問題発生の原因としては，次の四つがある.

i)　デバイスから応答がない

ii)　割込みのベクタ番号を取り込むことができない

iii)　メモリ管理ユニットに対する不正なアクセス要求

iv)　その他のアプリケーションに依存するエラー

バス・エラー信号とホルト信号は相互に作用し，エクセプション処理(exception processing)を実行するのか，現在のバス・サイクルを再実行するのかを決定する.

(2) リセット($\overline{\text{RESET}}$)

リセット信号線から入力される外部リセット信号によってMPUはリセットされ，システム・イニシャライズを開始する．RESET命令によってリセット信号が出力された場合は，これに接続されている外部デバイスがリセットされる(プロセッサの内部ステータスは変化しない).

外部ホルト信号およびリセット信号が同時に入力されると，MPUおよび外部デバイスの全システムがリセットされる.

(3) ホールト($\overline{\text{HALT}}$)

外部デバイスがこのホールト信号線を“low”にすると，現在動作中のバス・サイクルが終了した時点でMPUは停止する．この入力を使ってMPUがホールトされた場合，すべての制御信号はインアクティブとなり，すべてのスリー・ステートの信号線はハイ・インピーダンス状態となる.

2重バス障害のような状態でMPUが命令の実行を停止した場合，MPUはこのホルト信号をアクティブにしてMPUが停止したことを外部デバイスに連絡する.

2.5.7 6800の周辺制御

6800周辺制御信号を使うことによって，同期式の6800周辺デバイスと非同期式の68000とのインタフェースを行なうことができる.

(1) イネーブル(E)

この信号は，すべての6800タイプの周辺デバイスに共通な標準イネーブル信号である．この出力の1サイクルは68000の10クロックに相当する(6クロックが“low”，4クロックが“high”).

(2) バリッド・メモリ・アドレス($\overline{\text{VMA}}$)

この信号の出力により，6800周辺デバイスは，アドレス・バス上にあるアドレスが有効であり，MPUがイネーブル信号と同期して動作していることを知ることができる.

この信号は，周辺デバイスが6800タイプのデバイスであることを示すバリッド周辺アドレス($\overline{\text{VPA}}$)入力に対してのみアクティブとなる.

(3)　バリッド・ペリフェラル・アドレス($\overline{VPA}$)

この入力信号は，アドレスされたデバイスあるいは領域が6800周辺デバイスのアドレスであり，データ転送がイネーブル(E)信号と同期して行なわれることを示す信号である．また，割込み処理に当たって，MPUがオート・ベクタリングを行なうように指示するのにも使われる．

2.5.8　MPUステータス(ファンクション・コードFC0, FC1, FC2)

ファンクション・コード出力信号は，現在実行中のMPUのモード(スーパバイザあるいはユーザ)とメモリの参照区分を示している(表2.16)．

表2.16　ファンクション・コード出力

FC2	FC1	FC0	参照区分
low	low	low	(未定義)
low	low	high	ユーザ・データ
low	high	low	ユーザ・プログラム
low	high	high	(未定義)
high	low	low	(未定義)
high	low	high	スーパバイザ・データ
high	high	low	スーパバイザ・プログラム
high	high	high	割込みアクノリッジ

ファンクション・コードがプログラム参照を示すのは次の四つの場合である．

i)　アドレス・ソースがプログラム・カウンタの場合

ii)　リセット・ベクタのフェッチ・サイクルの場合

また，ファンクション・コードがデータ参照を示すのは次の場合である．

i)　アドレス・ソースがプログラム・カウンタ以外のオペランド・リード・サイクルの場合

ii)　オペランド・ライト・サイクルの場合

iii)　リセット以外のベクタ・フェッチ・サイクルの場合

ファンクション・コードの出力を利用して，アドレス空間をスーパバイザ・プログラム，スーパバイザ・データ，ユーザ・プログラム，ユーザ・データの四つの空間に分離することによって，メモリ空間を最大64Mバイトまで拡張

することもできる.

ファンクション・コード出力信号は，アドレス・ストローブ($\overline{AS}$)がアクティブのときは常に有効である.

2.5.9 クロック(CLK)

クロックはTTL (transistor transistor logic)コンパチブルな信号で，入力された信号は内部でバッファされ，MPUが必要とする内部クロックが生成される.

2.5.10 68000の命令のプリフェッチ(pre-fetch)

8086には6バイトの命令キューがあり，命令のプリフェッチ(先読み)を行なっていることはよく知られているが，68000が2ワード(4バイト)の命令キューをもち，命令の先読みを行なっていることはほとんど知られていない.これは，8086の命令の先読みがマニュアルや雑誌などでたびたび紹介されているのに対し，68000のそれは，マニュアルなどにもほとんど記載されていないからである.

68000の命令の先読みを簡単に説明すると，だいたい次のようになる.

まず，命令の解読，実行が開始された場合，命令のオペレーション・ワードとそれに続くワードの2ワードが命令キューに取り込まれる.命令キューに取り込まれたオペレーション・ワードは，インストラクション・レジスタにロードされ，解読される.後に続くワードは，先に命令キューに取り込まれたワードが使用済みになるに伴い，ワード単位で順にメモリから命令キューへ取り込まれていく.そして，一つの命令の最後のワードが命令キューから取り出されて実行されるときには，すでに命令キューに取り込まれている次の命令のオペレーション・ワードの解読が開始される.

ブランチ命令やジャンプ命令の場合には，命令キューは無効になり，すでに命令キューに取り込んだ内容は無視される.

2.6 エクセプション処理

2.6.1 処理状態

68000には三つの処理状態がある.三つの処理状態とは，ノーマル(normal)状態，エクセプション(exception)状態，ホールト(halted)状態の三つで，

68000は必ずどれか一つの処理状態をとる．

ノーマル処理状態は，命令を実行しているごく普通の状態で，メモリを参照して命令とオペランドをフェッチし，結果を再びメモリに格納するといったいわゆる通常の動作状態を指す．ノーマル処理状態の特別な場合として，MPUがSTOP命令を実行して停止状態にある場合が含まれる．

エクセプション処理状態は，割込み，トラップ命令，トレースおよびその他のエクセプションと関係があり，エクセプションが発生することによりMPUはエクセプション処理状態となる．エクセプションは，内部的には命令または命令実行中に引き起こされる異常によって，外部的には割込み，バス・エラーあるいはリセットによって発生する．

ホールト処理状態は，ハードウェアに重大な故障があり動作不能であることを示す．たとえば，バス・エラーのエクセプション処理中に他のバス・エラーが発生した場合には，MPUは，システムが使用不能であるとみなしてMPUをホールト状態とする．ホールト状態となったMPUのリストートは，外部からのリセットによって(のみ)可能となる．

2.6.2　プリビリッジ状態†

MPUには，二つのプリビリッジ状態(privilege state)——スーパバイザ状態(supervisor state)とユーザ状態(user state)がある．MPUは，この二つのプリビリッジ状態のうちのどちらか一つの状態で動作する．

このプリビリッジの機構を利用して，コンピュータ・システムの安全性を高めることができる．すなわち，オペレーティング・システムはスーパバイザ状態で実行してすべてのリソースをアクセス可能にし，オペレーティング・システムを除く他のプログラムはユーザ状態で実行して，アクセスできるリソースを限定するようにする．これにより，ユーザ・プログラムのエラーに起因するコンピュータ・システムへの影響を最小にすることができる．

(1)　スーパバイザ状態

MPUのプリビリッジ状態は，ステータス・レジスタのSビット(ビット13)で決まる．すなわち，Sビットが“1”の場合にMPUはスーパバイザ状態となる．スーパバイザ状態はユーザ状態よりもレベルが高く，スーパバイザ状態では68000のすべての命令を実行することができる．命令を実行したときのファ

† 特権状態という場合もある．

ンクション・コードの出力はスーパバイザ参照となる．MPU がスーパバイザ状態のときには，システム・スタック・ポインタを使用する命令も，アドレス・レジスタ A7 を使用する命令も，両方ともスーパバイザ・スタック・ポインタをアクセスすることになる．

Sビットの状態とは無関係に，すべてのエクセプション処理はスーパバイザ状態で実行される．また，エクセプション処理中のファンクション・コードの出力は，スーパバイザ参照となる．エクセプション処理中のすべてのスタック・オペレーションは，スーパバイザ・スタック・ポインタを使用して行なわれる．

(2) ユーザ状態

命令の実行に当たって，ステータス・レジスタのSビットが調べられる．このSビットが“0”であれば，MPU はユーザ状態で命令を実行する．

大部分の命令はユーザ状態でも実行できるが，システムに重大な影響を及ぼすいくつかの命令は特権命令 (privileged instruction) となっており，ユーザ状態では特権命令を実行することはできない．この意味で，ユーザ状態はスーパバイザ状態よりも低いレベルのプリビリッジ状態ということになる．特権命令には，STOP 命令，RESET 命令，ステータス・レジスタのシステム・バイトを変更する命令，“MOVE to USP”命令，”MOVE from USP”命令がある．ユーザ状態で特権命令を実行すると，特権違反のエクセプションが発生する．

ユーザ状態で命令を実行したときのファンクション・コードの出力は，ユーザ参照となる．また，MPU がユーザ状態のときには，システム・スタック・ポインタを使用する命令も，アドレス・レジスタ A7 を使用する命令も，両方ともユーザ・スタック・ポインタをアクセスすることになる．

(3) プリビリッジ状態の変更

MPU がユーザ状態で命令を実行中のときは，エクセプション処理だけがユーザ状態をスーパバイザ状態へ変更することができる．エクセプション処理では，それまでのステータス・レジスタのSビットは退避され，Sビットは“1”にセットされる．したがって，指定されたアドレスからエクセプション処理の命令が実行されるときには，MPU はスーパバイザ状態となっている．

スーパバイザ状態からユーザ状態への変更は，“RTE”，“MOVE to SR”，“ANDI to SR”，“EORI to SR”の四つの命令によって可能となる．RTE

命令では，スーパバイザ・スタックから（退避中の）ステータス・レジスタとプログラム・カウンタの内容がフェッチされてそれぞれのレジスタにロードされ，新しいプリビリッジ状態が指定される．そして，新しいプリビリッジ状態のもとで新しいプログラム・カウンタのアドレスから命令の実行が開始される．また，MOVE，ANDI，EORI の各命令では，それぞれの命令のオペランドの内容に従ってステータス・レジスタの内容が更新され，新しいプリビリッジ状態がセットされる．

以上をまとめて図 2.10 に示す．

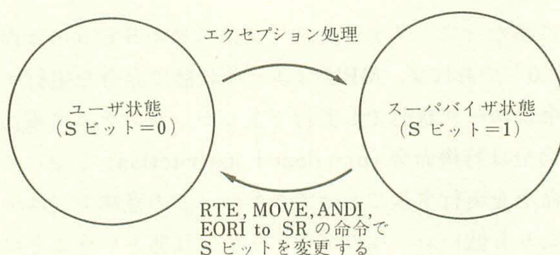


図 2.10　スーパバイザ状態/ユーザ状態の遷移

2.6.3　エクセプション処理

(1)　エクセプション・ベクタ†

表 2.17 に，エクセプション・ベクタの割当てを示す．エクセプション・ベクタの長さは，リセット・ベクタの 4 ワード（8 バイト）を除いてすべて 2 ワード（4 バイト）である（図 2.11）．リセット・ベクタの場合は，プログラム・カウンタにセットするアドレス（リセット・ルーチンの先頭アドレス）のほかに，システム・スタック・ポインタにセットするアドレスもエクセプション・ベクタとなるためである．

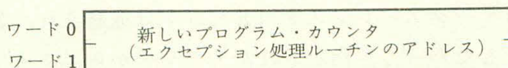


図 2.11　エクセプション・ベクタ

† 例外ベクタともいう．

表 2.17 エクセプション・ベクタの割当て

ベクタ番号	アドレス 10進	アドレス 16進	参照区分	割当て
0	0	000	SP	リセット：システム・スタック・ポインタの初期値
	4	004	SP	リセット：プログラム・カウンタの初期値
2	8	008	SD	バス・エラー
3	12	00C	SD	アドレス・エラー
4	16	010	SD	不当命令
5	20	014	SD	ゼロによる割り算
6	24	018	SD	CHK 命令
7	28	01C	SD	TRAPV 命令
8	32	020	SD	特権違反
9	36	024	SD	トレース
10	40	028	SD	1010 エミュレータ
11	44	02C	SD	1111 エミュレータ
12*	48	030	SD	(未定義)
13*	52	034	SD	(未定義)
14*	56	038	SD	(未定義)
15*	60	03C	SD	(未定義)
16～23*	64 95	040 05F	SD	(未定義：040から05Fまで)
24	96	060	SD	スプリアス割込み
25	100	064	SD	レベル1割込みオート・ベクタ
26	104	068	SD	レベル2割込みオート・ベクタ
27	108	06C	SD	レベル3割込みオート・ベクタ
28	112	070	SD	レベル4割込みオート・ベクタ
29	116	074	SD	レベル5割込みオート・ベクタ
30	120	078	SD	レベル6割込みオート・ベクタ
31	124	07C	SD	レベル7割込みオート・ベクタ
32～47	128 191	080 0BF	SD	TRAP 命令ベクタ
48～63*	192 255	0C0 0FF	SD	(未定義：0C0から0FFまで)
64～255	256 1023	100 3FF	SD	ユーザ割込みベクタとして100～3FFを使える

SP：スーパバイザ・プログラム，SD：スーパバイザ・データ

＊ ベクタ番号12～23，48～63は将来使用される可能性があるので，ユーザの周辺デバイスにこれらの番号を割り当ててはいけない

エクセプション・ベクタの番号は8ビットで，各エクセプションの要因に基づいて内部あるいは外部で生成される．たとえば，割込みの場合，インタラプト・アクノリッジ・バス・サイクル中に周辺デバイスがデータ・バス・ラインD0～D7に8ビットのベクタ番号（図2.12）をのせ，MPUに送る．MPUは，このベクタ番号を4倍して24ビットのアドレス（図2.13）に変換し，エクセプション・ベクタのアドレスとする．

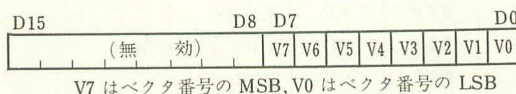


図 2.12　周辺デバイスから与えられるベクタ番号

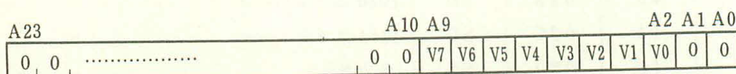


図 2.13　8ビット・ベクタ番号から変換されるアドレス

表2.17に示すように，エクセプション・ベクタとして全部で512ワード（1024バイト）ありメモリの0番地から1023番地に割り当てられている．ベクタ番号0～63がシステム用で64～255がユーザ割込みベクタ用になっているが，0～63のベクタをユーザ割込みベクタとして重複して使用することも可能である．

(2)　エクセプションの種類

エクセプションは，外部あるいは内部の種々の要因により発生する．外部要因により発生するエクセプションとしては，割込み，バス・エラー，リセットがある．割込みは，周辺デバイスから出されるMPUの動作要求であり，一方，バス・エラーとリセットはアクセス制御とMPUのリスタートに使用される．内部要因により発生するエクセプションとしては，命令によるものや，アドレス・エラー，トレースによるものがある．エクセプションを発生させる命令としては，TRAP，TRAPV，CHK，DIVS，DIVUがあり，その他，奇数番地からのワード・アクセスによるアドレス・エラーや不当命令，特権命令違反によってもエクセプションが発生する．

(3) エクセプション処理シーケンス

エクセプション処理には，次の四つのステップがある．第1ステップでは，ステータス・レジスタの一時的なコピーが作られ，その後でステータス・レジスタのSビットが“1”にセットされ，MPUはスーパバイザ状態になる．同時にTビットも“0”となり，トレースは無効となる．リセットと割込みによるエクセプションの場合には，ステータス・レジスタの割込みマスクも更新される．

第2ステップでは，エクセプションのベクタ番号が決定される．割込みの場合には，ベクタ番号をMPUがフェッチし（割込みアクノリッジ），その他のエクセプションの場合には，内部ロジックによりベクタ番号が決定される．このベクタ番号を使って対応するエクセプション・ベクタのアドレスが作成される．

第3ステップでは，リセットの場合を除いて，現在のプログラム・カウンタの値と退避されたステータス・レジスタのコピーがスーパバイザ・スタックにプッシュされる．スタックにプッシュされるプログラム・カウンタの値は，通常はまだ実行されていない次の命令のアドレスを指しているが，バス・エラーとアドレス・エラーの場合には，プログラム・カウンタの値を特定することは不可能である（エラーを引き起こした命令のアドレスよりも先に進んでいる可能性がある）．また，バス・エラーとアドレス・エラーのエクセプションの場合には，実行中のMPUの内容を示すさらに詳しい情報がスタックにプッシュされる．

最後の第4ステップでは，新しいプログラム・カウンタの値がエクセプション・ベクタからフェッチされ，MPUはプログラム・カウンタの示すエクセプション処理ルーチンの先頭から通常の命令の実行を再開する．

図2.14に，リセットを除くエクセプション処理のシーケンスを示す．

(4) 多重エクセプション

ここでは，同時に複数のエクセプションが発生した場合の処理について説明する．

エクセプションは，その発生要因と優先度に応じて表2.18に示すような三つのグループに分類される．三つのグループの中では，グループ0のエクセプションが最も優先度が高く，グループ2のエクセプションの優先度が最も低く

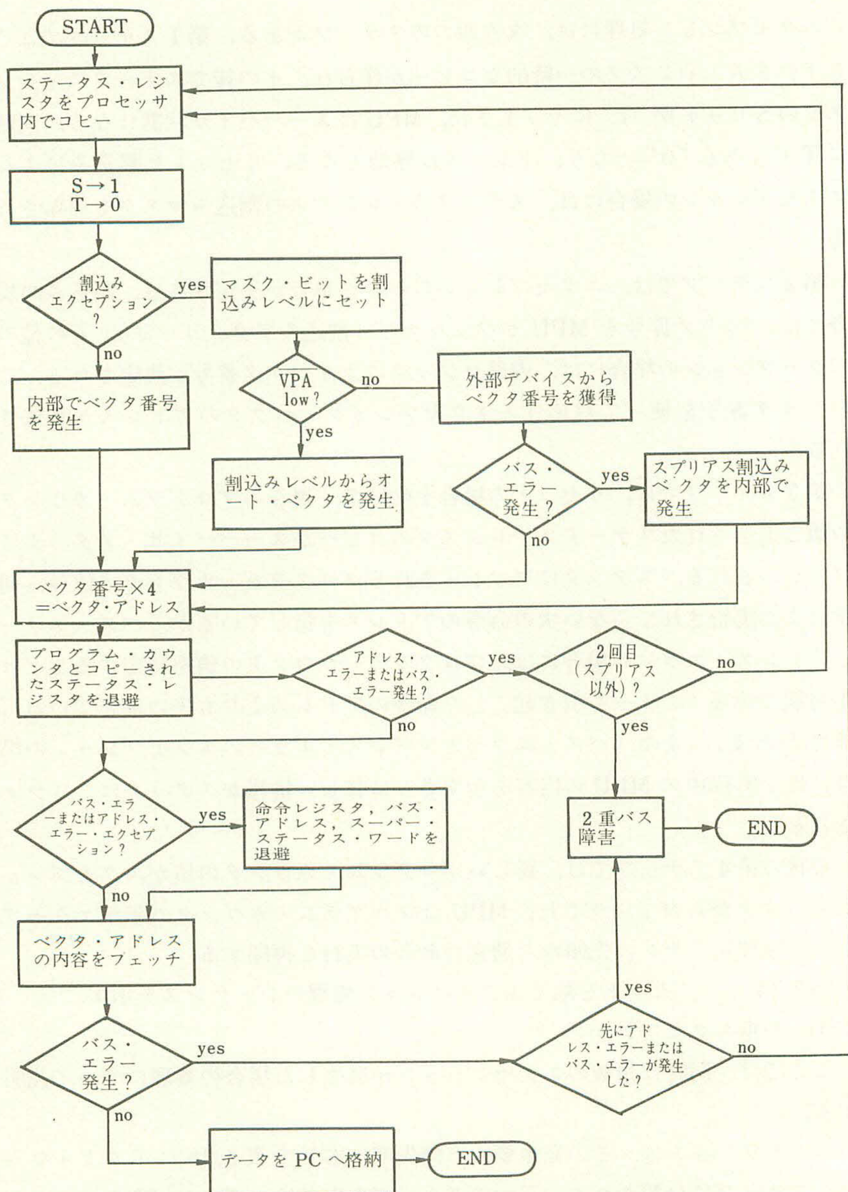


図 2.14　エクセプション処理シーケンス（リセットの場合を除く）

表 2.18 エクセプション処理の分類と優先度

グループ	エクセプション	処 理	優先度
0	リセット バス・エラー アドレス・エラー	エクセプション処理は，次のマイナー・サイクル (minor cycle) から始まる	高 ↑
1	トレース 割込み 不当命令 特権命令違反	エクセプション処理は，次の命令の前に始められる	│
2	TRAP, TRAPV, CHK, 0による割り算	エクセプション処理は，命令の通常の実行により始まる	↓ 低

なっている．グループ0の中では，リセットが最も優先度が高く，次にバス・エラー，アドレス・エラーの順になっている．グループ1の中では，トレースが割込みよりも優先度が高く，割込みは不当命令，特権命令違反よりも優先度が高くなっている．グループ2の中では1度に一つの命令しか実行できないため優先順位はない．

二つのエクセプションが同時に発生した場合，どちらのエクセプション処理を最初に行なうかは両者の優先順位によって決まる．たとえば，TRAP命令の実行中にバス・エラーが発生した場合には，バス・エラーの処理が先に行なわれ，TRAP命令の処理は中断される．また，Tビットが“オン”の状態で命令実行中に割込み要求が発生した場合は，トレース・エクセプションの処理が優先して最初に処理され，次に割込みエクセプションの処理が行なわれる．

2.6.4 エクセプション処理の詳細

以下では，個々のエクセプションの発生要因およびその処理内容について詳しく説明する．

(1) リセット

リセット (reset) は最も優先度の高いエクセプションである．リセット信号は，システムのイニシャライズや重大な障害からの回復を目的に設けられている．リセット信号が入力されると，進行中の処理はすべて強制的に停止され，打ち切られる．ステータス・レジスタのSビットはスーパバイザ状態にセットされ，トレース・モードのTビットはオフとなる．割込みマスクはレベル7に

セットされる．ベクタ番号0がMPU内部で生成され，メモリ（スーパバイザ・プログラム領域）のアドレス0番地にセットされているリセット・エクセプション・ベクタがフェッチされる．リセット・エクセプションの場合には，プログラム・カウンタとステータス・レジスタの退避は行なわれない．

リセット・エクセプション・ベクタの始めの2ワードは，スーパバイザ・スタック・ポインタのアドレスとしてフェッチされ，残りの2ワードは，プログラム・カウンタにセットするアドレスとしてフェッチされる．プログラム・カウンタにフェッチしたアドレスがセットされると，“リセット・ルーチン（reset routine）”の命令の実行が開始される．プログラム・カウンタにセットするアドレスは，パワー・オン/リスタートなどの処理を行なうリセット・ルーチンの先頭アドレスを指定しておけばよい．

RESET命令では，リセット・エクセプション・ベクタのフェッチは行なわれず，外部デバイスのリセットだけが行なわれる．このように，ソフトウェアによってシステムを所定の状態にリセットしてから処理を続けることもできる．

図2.15に，リセット・エクセプション処理シーケンスを示す．

(2)　割込み

割込み（interrupt）には，七つの優先レベルがある．デバイスは割込み優先レベルの範囲内であれば数に制限はなく，いくらでも接続することができる．

優先レベルは1から7まであり，レベル7が最も優先度が高くなっている．ステータス・レジスタに現在のMPUの優先レベルを示す3ビットの割込みマスク（ビット8～10）があり，現在のMPUの優先レベル以下の割込みはすべて割込みが抑止される．

割込み制御信号（$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$）の割込みレベルをエンコードすることによって，割込み要求がMPUに伝えられる．割込みレベル0は，割込み要求がないことを意味している．

MPUは，割込み要求を受け取ってもすぐにエクセプション処理は実行せず，割込み要求を待機状態（pending）とする．待機させられた割込み要求は各命令の実行と実行の間に受け付けられ，優先レベルの比較が行なわれる．割込みレベルが現在のMPUの優先レベル以下の場合には，現在の命令の処理が継続し

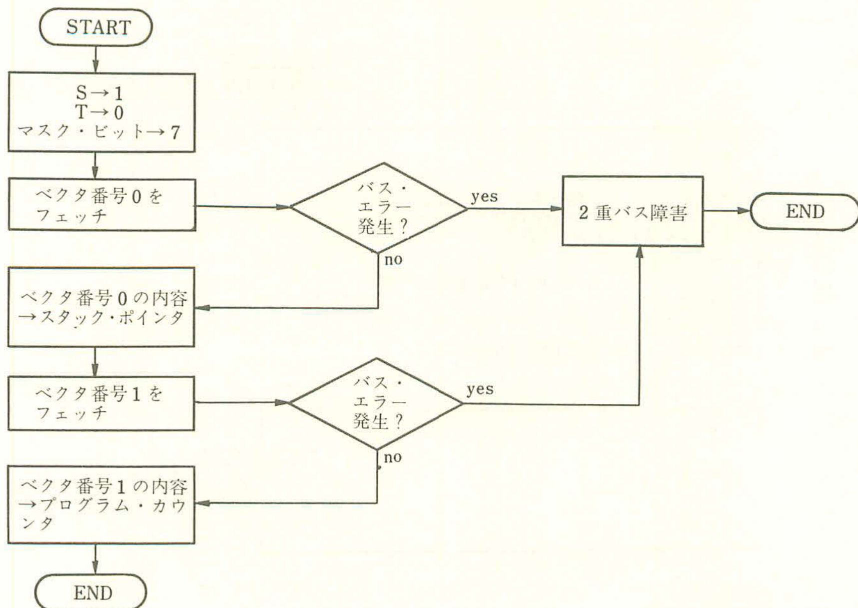


図 2.15　リセット・エクセプション処理シーケンス

で実行され，割込みエクセプションの処理は実行待ち状態となる．

割込みレベルが現在のMPUの優先レベルよりも高い場合は，割込みエクセプションの処理が開始される．まず最初にステータス・レジスタのコピーが退避され，Sビットが“1”にセットされてスーパバイザ状態となる．同時にトレース・モードのTビットがリセットされ，割込みマスク・ビットの優先レベルは受け付けた割込みの優先レベルにセットされる．MPUは，割込みをかけたデバイスから送出されたベクタ番号をフェッチする．このときの参照区分は割込みアクノリッジ参照で，受け付けた割込み優先レベルがアドレス・バス上に出力される．図2.16に割込みアクノリッジ・シーケンスのフローチャートを示す．外部ロジックでオート・ベクタリングを要求している場合には，割込みの優先レベルに応じてMPU内部でベクタ番号25～31が生成される．外部ロジックがバス・エラーを示している場合，割込みは“スプリアス(spurious)”とみなされ，スプリアス割込みベクタを参照するベクタ番号24が生成される．

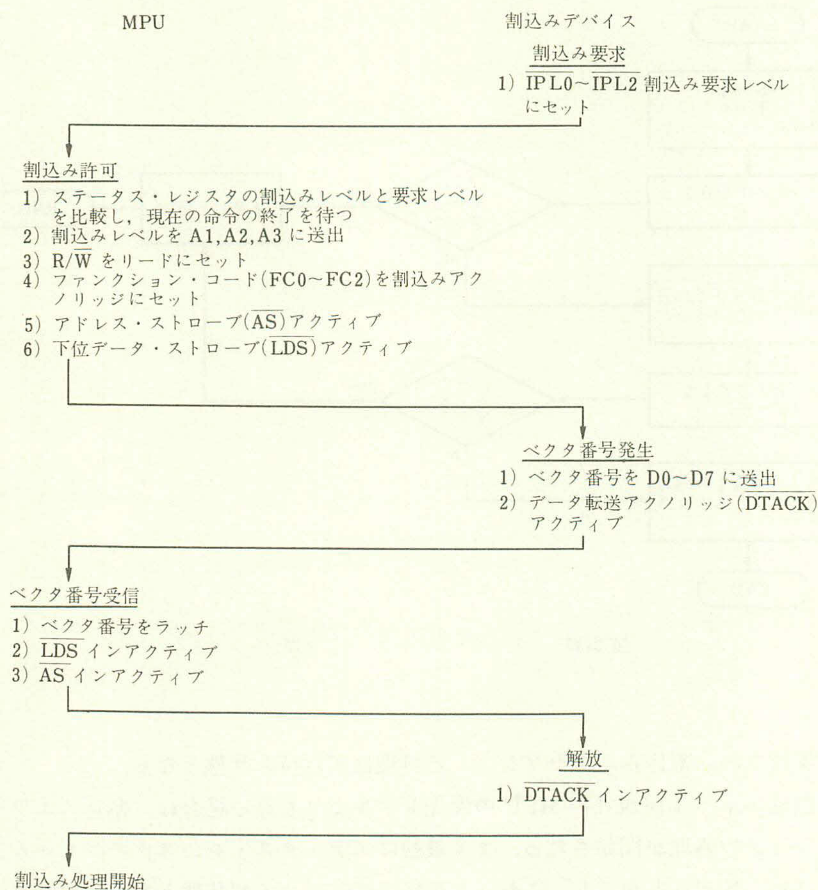


図 2.16　割込みアクノリッジ・シーケンス・フローチャート

それから，プログラム・カウンタとステータス・レジスタのコピーがスーパバイザ・スタックに退避される．退避されるプログラム・カウンタの値は，割込みがなかった場合に実行されていたはずの命令を指している．つづいて，ベクタ番号で示される割込みベクタの内容がフェッチされてプログラム・カウンタへロードされ，“割込み処理ルーチン（interrupt handling routine）”の命令が実行される．

優先レベル7の割込みは，レベル1〜6の場合と異なり，割込みマスク・ビットで割込みを禁止することはできない．この意味で，優先レベル7の割込み

は“ノン・マスカブルな割込み(non-maskable interrupt)”である．したがって，割込み要求レベルをレベル1～6からレベル7に変更するたびに割込みが発生することになる．

(3) トラップ

トラップ(trap)は命令の実行によって発生するエクセプションである．命令の実行中にMPUの異常を認めるか，あるいは，トラップを発生する命令を実行した場合にトラップが発生する．

トラップ・エクセプション処理では，はじめにステータス・レジスタがコピーされる．そして，Sビットがスーパバイザ状態にセットされ，トレース・モードのTビットはオフになる．トラップ・エクセプション・ベクタを参照するベクタ番号が，MPU内部で生成される．TRAP命令の場合は，ベクタ番号の一部は命令のオペレーション・ワードのビット0～3で与えられる．次に，プログラム・カウンタとステータス・レジスタのコピーがスーパバイザ・スタックに退避される．退避されるプログラム・カウンタの値は，トラップを発生させた命令の次の命令のアドレスを指している．最後に，エクセプション・ベクタにセットされているアドレスから命令の実行が開始される．

トラップを発生させる特別な命令がいくつかある．TRAP命令は常にエクセプションを発生させる命令で，ユーザ・プログラムにとってはシステム・コール(system call)を実現する上で有用な命令である．TRAPV, CHK命令は，ユーザ・プログラムが実行中のエラーを検出した場合にエクセプションを発生させる命令で，算術演算のオーバフローやデータの境界値オーバを検出した場合にエクセプションを発生させる．

DIVS (signed divide), DIVU (unsigned divide) 命令では，“0”で割り算を行なうとエクセプションが発生する．

(4) 不当命令と不実行命令

正常な命令のオペレーション・ワードのビット・パターンと異なるビット・パターンの命令を，“不当命令(illegal instruction)”と呼ぶ．命令実行中にこのような命令がフェッチされると，不当命令エクセプションが発生する．

命令のオペレーション・ワードのビット15から12までのパターンが“1010”と“1111”の命令は，特に“不実行命令(unimplemented instruction)”として区別され，これらのパターンに個別にエクセプション・ベクタが割り当てられ

ている．これにより効率の良いエミュレーション（emulation）が可能で，オペレーティング・システムはプログラム・エラーを検出したり，ソフトウェアで不実行命令をエミュレートすることができる．

不当命令と不実行命令のエクセプション処理は，トラップの場合と同じである．MPUが命令をフェッチ，デコードすることにより不当命令，不実行命令がみつかるとエクセプション処理が始まる．まずステータス・レジスタがコピーされる．次にSビットがスーパバイザ状態にセットされ，トレース・モードがオフになる．そして，不当命令，不実行命令に応じてそれぞれのエクセプション・ベクタを参照するベクタ番号4，10，11が生成される．現在のプログラム・カウンタとステータス・レジスタのコピーがスーパバイザ・スタックに退避され，エクセプション・ベクタにセットされていたアドレスから命令の実行が開始される．退避されたプログラム・カウンタの値は，不当命令または不実行命令のアドレスを指している．

(5)　特権違反

システムの安全性，信頼性を確保するためにいくつかの命令が特権命令となっている．ユーザ状態で特権命令を実行すると特権違反（privilege violation）のエクセプションが発生する．特権命令には次のような命令がある．

```
STOP
RESET
RTE
MOVE to SR
AND (word) immediate to SR
EOR (word) immediate to SR
OR  (word) immediate to SR
MOVE USP
```

特権違反のエクセプション処理は，不当命令のエクセプション処理に似ている．命令のフェッチ，デコードの後で，特権違反であればエクセプション処理が開始される．ステータス・レジスタがコピーされた後で，Sビットがスーパバイザ状態にセットされ，トレース・モードのTビットはオフになる．そして，特権違反のエクセプション・ベクタをロードするためのベクタ番号8が生成され，現在のプログラム・カウンタとステータス・レジスタのコピーがスーパバイザ・スタックに退避される．退避されるプログラム・カウンタの値は，特権違反を引き起こした命令の第1ワードのアドレスを示している．最後に，特権

違反のエクセプション・ベクタにセットされていたアドレスから命令の実行が開始される.

(6) トレース

68000にはトレース(tracing)機能があり，プログラムの開発に有効である.トレース・モードで命令を実行すると，各命令の実行後にトレース・エクセプションが発生し，テスト中のプログラムの実行をデバッギング・プログラムで簡単にモニタすることができる.

トレース機能は，ステータス・レジスタのTビット(ビット15)を使って実現される.Tビットがオフの場合はトレース機能は働かず，通常どおり命令を実行していく.命令の実行を開始する前にTビットがオンの場合，命令実行終了後トレース・エクセプションが発生する.割込み受付けのため命令が実行されなかった場合や，命令が不当命令あるいは特権違反の命令であった場合には，トレース・エクセプションは発生しない.また，リセット，バス・エラー，アドレス・エラーのエクセプションによって命令の実行が中断された場合も，トレース・エクセプションは発生しない.命令が実際に実行され，割込みが命令の実行終了まで待機させられている状態では，割込みエクセプションより先にトレース・エクセプションが処理される.また，命令実行中にその命令自身によってエクセプションが発生した場合は，そのエクセプション処理をトレース・エクセプションより先に行なう.上記処理の例として，トレース・モードがオンでTRAP命令実行中に割込みが発生した場合について考えてみる.この場合は，最初にトラップ・エクセプションが処理され，次にトレース・エクセプションが，そして最後に割込みが処理される.

トレースのエクセプション処理は非常に簡単である.命令の実行終了後，次の命令の実行を開始する前にトレース・エクセプション処理が始まる.まずステータス・レジスタのコピーが作られる.次にスーパバイザ状態にセットされ，トレース・モードがオフになりこれ以上のトレースは抑止される.トレース・エクセプション・ベクタ参照のためのベクタ番号9が生成され，プログラム・カウンタとステータス・レジスタのコピーがスーパバイザ・スタックに退避される.退避されるプログラム・カウンタの値は次の命令のアドレスを指している.そして，トレース・エクセプション・ベクタにセットされていたアドレスから“トレース・ルーチン(tracing routine)”の命令の実行を開始する.

(7)　バス・エラー

外部ロジックからバス・エラー(bus error)のエクセプション処理の要求が出されることによって，バス・エラー・エクセプションが発生する．MPUのバス・サイクルは中断され，MPUの処理は終了させられ，MPUはただちにバス・エラーのエクセプション処理を開始する．まずステータス・レジスタのコピーが作られる．その後でMPUの処理状態がスーパバイザ状態にセットされ，トレース・モード・ビットはオフとなる．ベクタ番号2が生成され，バス・エラー・ベクタが参照される．バス・エラーのエクセプション処理が要求された時点では，MPUはバス・エラーの原因となった命令のアドレスは指していないと考えられるので，MPUのより詳細な処理内容に関する情報が必要になる．プログラム・カウンタとステータス・レジスタのコピーのほか，より詳細な情報を示す付加情報がスーパバイザ・スタックに退避される．退避されるプログラム・カウンタの値は，バス・エラーを引き起こした命令の第1ワードよりも2～10バイト程度先に進んでいる可能性が大きい．次の命令をフェッチしている間にバス・エラーが発生した場合は，たとえそのバス・エラーを発生させた命令が“branch”，“jump”，“return”命令であっても，退避されたプログラム・カウンタの値は現在の命令の付近のアドレスを示している．付加情報としては，MPUの処理中だった命令のオペレーション・ワードの内部コピーと，中断されたバス・サイクルによってアクセスされていたアドレスが退避される．

さらに，バス・エラー時のアクセスに関する次の情報

i)　リードかライトか

ii)　命令実行中か否か

iii)　バス・エラーが発生したときのファンクション・コード出力の参照区分

の三つの情報が退避される．ii)のMPUが命令実行中というのは，MPUがノーマル状態の場合，またはグループ2のエクセプション処理の場合を指しており，グループ0あるいはグループ1のエクセプション処理の場合は，命令実行中にはならない．図2.17にスーパバイザ・スタックに退避される情報の構成を示す．

以上の情報は，バス・エラーの回復処理には不十分かもしれないが，ソフトウェアのエラー診断には有用な情報である．最後にMPUはバス・エラーのエ

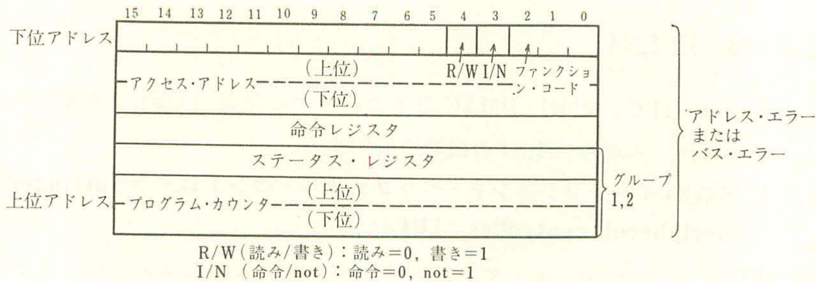


図 2.17　エクセプション処理のときの退避データ

クセプション・ベクタにセットされているアドレスから命令の処理を開始する．スーパバイザ・スタックにプッシュされた情報を基に処理をどこまで継続して実行するかは，ユーザのエラー処理ルーチンの自由に任される．

バス・エラーが，バス・エラー，アドレス・エラー，リセットのエクセプション処理中に発生した場合，MPU は停止し (halted)，処理はすべて打ち切られる．MPU はメモリの内容をすべて破壊することなくシステムから切り離されるので，上記の処理によって重大なシステムの故障を簡単に検出することができる．停止中の MPU のリスタートはリセット信号 ($\overline{\text{RESET}}$) によってのみ可能である．

(8)　アドレス・エラー

MPU がワード，ロング・ワードあるいは命令を奇数番地でアクセスしようとした場合，アドレス・エラー (address error) のエクセプションが発生する．アドレス・エラーのエクセプション処理は，バス・エラーの場合と同じで，バス・サイクルが中断され，MPU はその処理を中止してエクセプション処理を開始する．エクセプション処理の内容は，エクセプション・ベクタの番号(アドレス・エラーの場合は 3) を除いて，スーパバイザ・スタックに退避される情報も含め，バス・エラーの場合と同じである．また，バス・エラー，アドレス・エラー，リセットのエクセプション処理中にアドレス・エラーが発生すると MPU が停止するのも，バス・エラーの場合と同じである．

2.7 周 辺 LSI

68000には，IPC，MMU，DMACなどのいろいろな周辺LSIが準備されている．ここでは，それらのLSIの概略を説明する．

2.7.1 68120インテリジェント・ペリフェラル・コントローラ(intelligent peripheral controller：IPC)

68120 IPCは，汎用のユーザ・プログラマブルな入出力コントローラである．IPCは8ビットCPUを中心に，システム・インタフェース，シリアル・コミュニケーション・インタフェース，21本のパラレル入出力ライン，16ビット・タイマ，2048バイトROM，128バイトRAM，6個のセマフォ・レジスタ(semaphore register)から構成されており，八つの動作モードがある．RAM 128バイトと6個のセマフォ・レジスタは，内部CPUと外部プロセッサの両方からアクセスすることができ，外部からのアクセスはシステム・インタフェースを通して行なわれる．

以下に，IPCの特徴を示す．

(1) 68000および6809マイクロプロセッサとバス・コンパチブル
(2) 6800マイクロプロセッサおよび周辺LSIとバス・コンパチブル
(3) 8ビットCPU
(4) 2Kバイト ROM，128バイト・デュアル・ポート(dual-ported) RAM
(5) 6個のセマフォ・レジスタ
(6) 16ビット・タイマ
(7) 21本のパラレル入出力ラインおよび2本のハンド・シェイク・ライン
(8) シリアル・コミュニケーション・インタフェース
(9) 割込み機能，DMA機能
(10) ソフトウェアは6800，6801，6802および6803マイクロプロセッサと上方への互換性がある(upward compatible)
(11) ソースおよびオブジェクト・コードは6800，6801マイクロプロセッサと上方への互換性がある．
(12) 8×8ビットの乗算可能
(13) ソフトウェアによる制御

- セマフォ・レジスタ
- 16ビット・タイマ
- シリアル・コミュニケーション・インタフェース
- パラレル入出力ポート
- 割込み

(14) 5ボルト単一電源

図2.18にIPCのピン配置，図2.19にブロック図を示す．

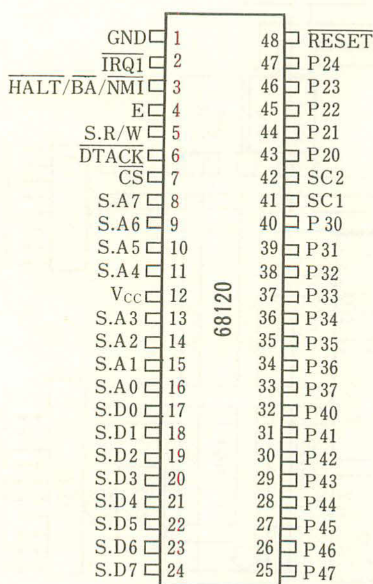


図 2.18　IPCピン配置

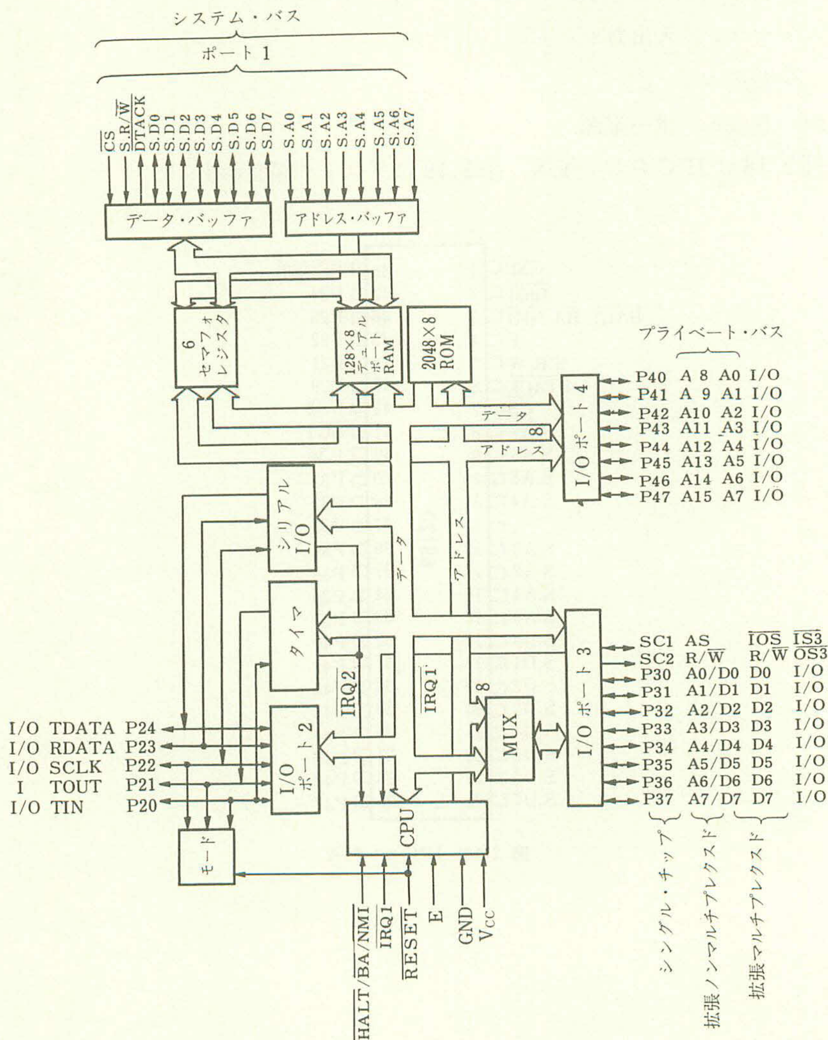


図 2.19　IPCブロック図

2.7.2 68122クラスタ・ターミナル・コントローラ(cluster terminal controller : CTC)

68122CTCは，ホスト・プロセッサの代わりに端末装置間のコミュニケーションを調整する働きがある．CTCは，標準的なプロトコル(protocol)で動作する端末装置のほかにも，ライン・プリンタのような出力専用の端末装置や，入力専用の端末装置のサポートも可能である．

CTCの特徴には，次のようなものがある．

(1) ホスト・プロセッサから全機能制御可能
(2) ブロードキャスト(broadcast)機能で，すべての端末装置に対するテキスト(text)の自動生成が可能
(3) ホスト・プロセッサからCTCのローカル・メモリの任意の部分のテストや変更が可能
(4) ステータス報告機能
(5) エコー制御(echo control)
(6) エラーの自動回復機能
(7) 5ボルト単一電源

図2.20に，CTCの接続概念図を示す．

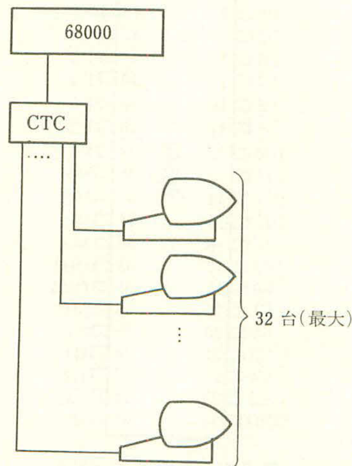


図 2.20　CTCによる端末装置の接続概念図

2.7.3　68540エラー・ディテクション・アンド・コレクション・サーキット (error detection and correction circuit：EDCC)

68540 EDCCは，8ビットおよび16ビットのデータ・バス上のデータをチェック/修正する機能を有しており，32ビットへの拡張も容易に行なうことができる．EDCCには内部診断レジスタ (internal diagnostic register) が内蔵されており，診断処理モード (diagnostic operating mode) でシステム診断テストやエラーのロギング (logging)† を行なうことができる．

EDCCには，次のような特徴がある．

(1)　EDCC 1個で8ビットおよび16ビットのサポート

(2)　EDCC 2個による32ビット・データのサポート

(3)　1ビット・エラー修正，2ビット・エラー検出

(4)　オール“1”またはオール“0”の致命的エラーの検出

(5)　パリティ・サポート (リードおよびライト)

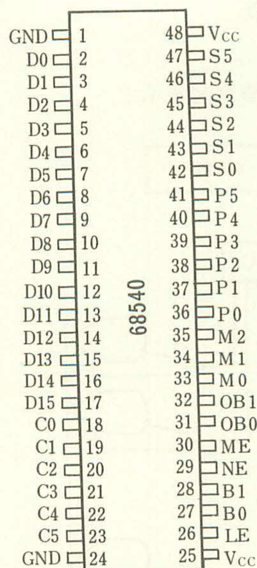


図 2.21　EDCCピン配置

† エラーの記録．

(6) エラー・フラグ

(7) 柔軟性のある診断処理モード

(8) 内部診断レジスタ内蔵

(9) 入出力データのラッチ(latch)

図2.21にEDCCのピン配置を，図2.22に接続概念図を示す.

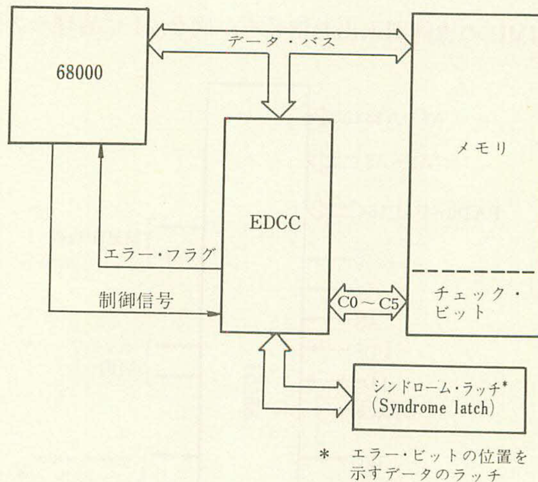


図 2.22 EDCCの接続概念図

2.7.4 68451メモリ・マネージメント・ユニット(memory management unit: MMU)

68451MMUは，68000の16Mバイトのアドレス空間のアドレス変換とメモリ・プロテクションを行なう. MMUはバス・マスタによってアクセスされ，バス・マスタから出力されるアドレスおよびファンクション・コードに基づいて，アドレス変換とメモリ・プロテクションを行なう.

MMUには，次のような特徴がある.

(1) アドレス空間のスーパバイザ領域とユーザ領域への分離

(2) ライト・プロテクションによるシステムの信頼性の向上

(3) 効率の良いメモリ・アロケーション(memory allocation)

(4) メモリの共有により内部プロセスのコミュニケーションができる

(5) アドレス空間に関するプログラミングの簡略化

(6)　複数のMMUでシステムを構成することもできる

(7)　ページング(paging)とセグメンテーション(segmentation)の両方をサポートする

(8)　32セグメントの管理が可能で，各セグメントのサイズは任意である

(9)　仮想メモリ(virtual memory)サポート可能

(10)　68000バス・コンパチブル

図2.23にMMUの機能別入出力信号を，図2.24に論理セグメントのマッピング例を示す．

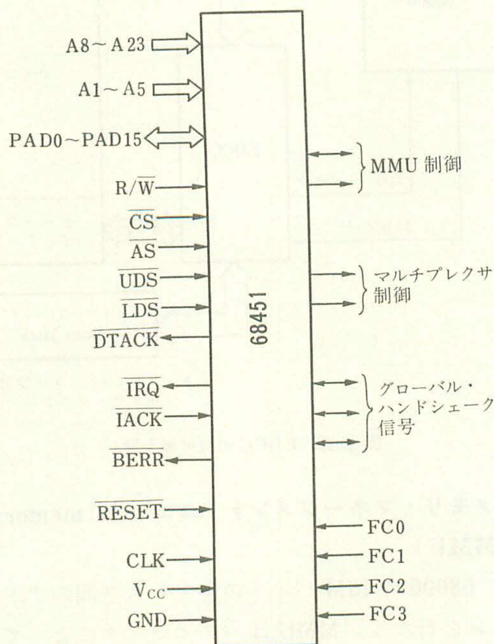


図 2.23　MMU機能別入出力信号

2.7.5　68450ダイレクト・メモリ・アクセス・コントローラ(direct memory access controller：DMAC)

68450 DMACにより，効率の良い柔軟性のあるデータ転送が可能で，チェイニング(chaining technique)やメモリ間ブロック転送により最適なデータ転送を行なうことができる．

以下に，DMACの特徴を示す．

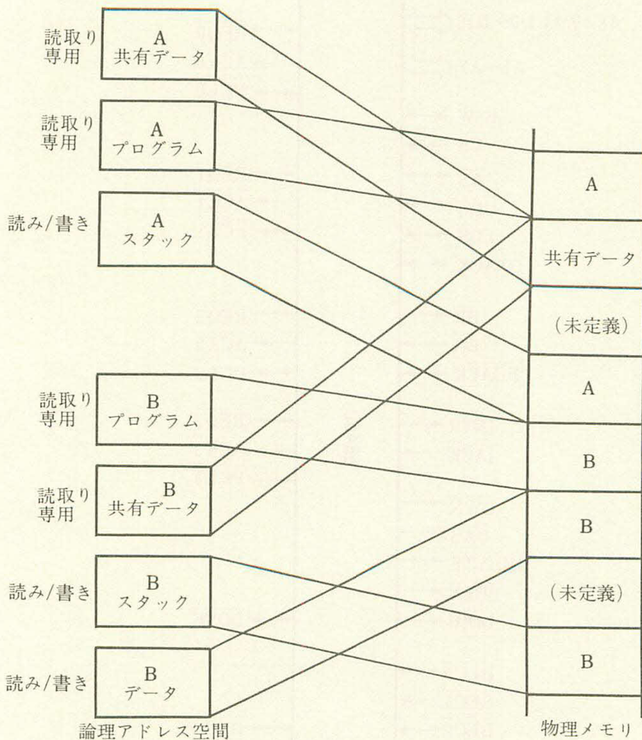


図 2.24　論理セグメントのマッピング

(1) 68000ファミリおよび6800周辺LSIとコンパチブル
(2) 独立な4組のチャネル(channel)
(3) バイト，ワードおよびロング・ワードの各データの転送が可能
(4) メモリ間ブロック転送
(5) チェインド(chained)およびアンチェインド(unchained)なデータの転送のサポート
(6) データ転送速度は最高4Mバイト/秒
(7) 割込みのベクタリングのサポート
(8) プログラマブルな優先度(priority)

図2.25にDMACの機能別入出力信号を，図2.26にDMAC内部のチャネルのレジスタの構成を示す.

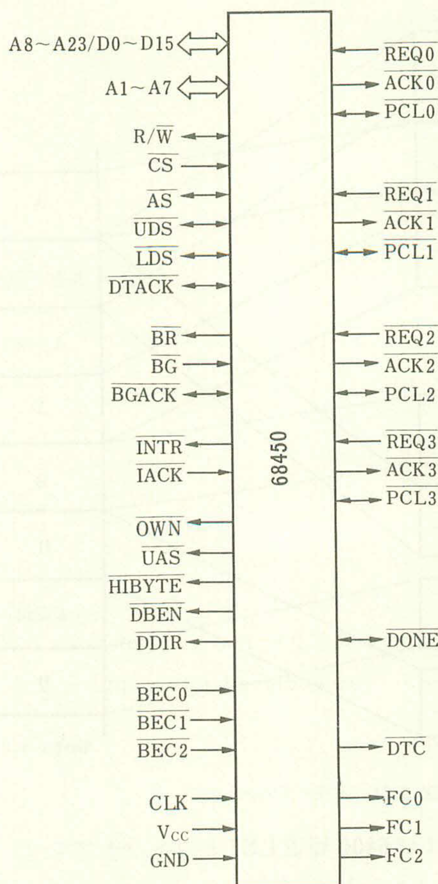


図 2.25　DMAC 機能別入出力信号

2.7.6　68230 パラレル・インタフェース/タイマ（parallel interface/timer : PI/T）

68230 PI/T は，プログラマブルな汎用パラレル・インタフェース用デバイスで，I/O 割込みとタイマの機能を備えている．PI/T は，周辺装置の要求に応じて 68000 に割込み要求を出したり，68450 DMAC に対して DMA 要求を出したりするようにプログラミング可能である．68000, DMAC に PI/T を接続する場合，外部ロジックは不要である．

PI/T には，二つの多重モード・ダブル・バッファド（multi-mode double-

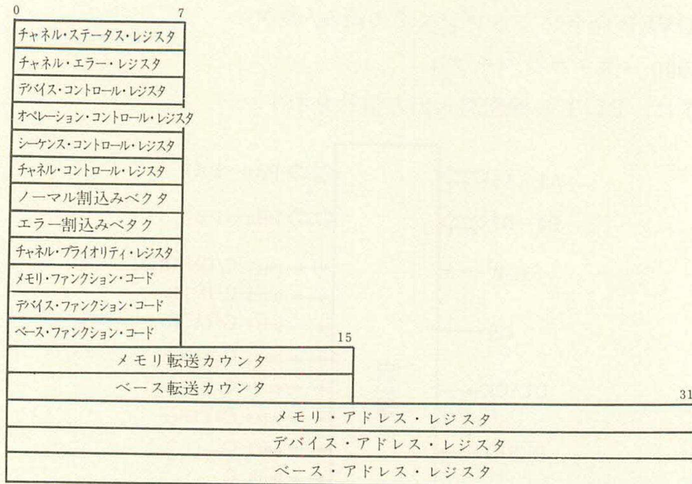


図 2.26　チャネルのレジスタ構成

buffered) I/O ポート（ポートAとポートB），8ビット I/O ポート，それに24ビットのプログラマブル・タイマのほか，個別の割込みベクタを発生させる優先度付き割込み要求発生用のロジック（prioritization logic）が含まれている．PI/T では自動ベクタ割込み（autovectored interrupt）もサポートできる．

以下に，PI/T の特徴を示す．

(1)　24本のプログラマブル入出力ライン
(2)　ポート・モード
- ビット・モード
- 単方向8ビット・モード
- 単方向16ビット・モード
- 両方向8ビット・モード
- 両方向16ビット・モード

(3)　24ビットのプログラマブル・タイマ
(4)　数種のタイマ・モード
(5)　割込み優先度付きロジックを内蔵
(6)　各要因に個別の割込みベクタ
(7)　割込みサービス要求と DMA サービス要求のサポート
(8)　ポートおよびタイマ・ステータス・レジスタを内蔵

(9) MOVEP 命令によるレジスタの読み/書き

(10) 68000バス・コンパチブル

図2.27に，PI/Tの機能別入出力信号を示す．

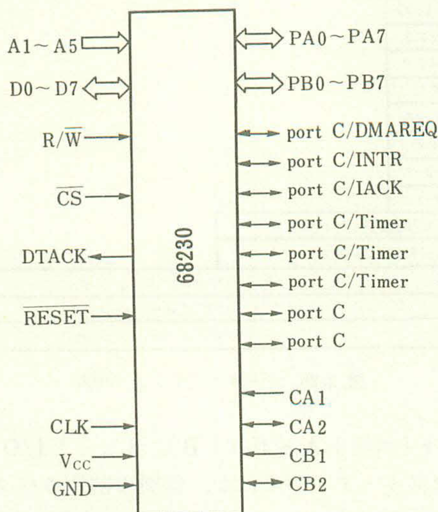


図 2.27　PI/T機能別入出力信号

2.7.7　68561マルチプロトコル・コミュニケーション・コントローラ（multi-protocol communication controller：MPCC）

68561 MPCCは，68000ファミリ用のシリアル・データ・コミュニケーション・インタフェースである．MPCCは通常の非同期転送のほか，ビットおよびバイト指向の同期転送の伝送手順（protocol）を扱うことができる．

MPCCには，次のような特徴がある．

(1) 次の伝送手順のサポート
- 非同期転送
- ビット指向同期転送（X, 25, SDLC, HDLC）
- バイト指向同期転送（BISYNC, DDCMP）

(2) 68000との直接インタフェース

(3) MPUインタフェースは68450 DMACとコンパチブル

(4) セルフ・テスト・ループ・モード（self test loop mode）

(5) 全二重もしくは半二重の動作

(6) CRC (cyclic redundancy check) キャラクタ生成およびエラー検出

(7) 水晶発振器およびボー・レート発生器 (baud rate generator) 内蔵

(8) ステータス報告機能

(9) バッファ用送信/受信レジスタ内蔵

(10) ベクタ割込みのサポート

(11) 68000 バス・コンパチブル

(12) 5 ボルト単一電源

図 2.28 に，MPCC の接続概念図を示す.

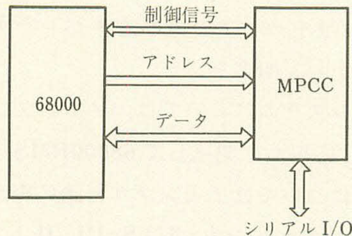


図 2.28　MPCC 接続概念図

参 考 書

1) Motorola Semiconductor Products Inc.：“*MC* 68000 16-*Bit Microprocessor User's Manual*”, September (1979).

2) 日立製作所：“日立マイクロコンピュータ・システム　HD68000 MPU (Micro Processing Unit)——暫定仕様——”.

3) Motorola Semiconductor Products Inc.：“*MC* 68000 *Advance Information* (*ADI*-814)” (1979).

4) Motorola Semiconductor Products Inc.：“*MC* 68000 *Course Notes* (*Technical Training*)”, January (1980).

5) 大渕竜太郎訳：「68000」, bit 別冊, “16 ビット・マイクロプロセッサ”, p. 173～222, 5 月, 共立出版 (1982).

6) 藤岡　旭：「68000 マイクロプロセッサの動作とハードウェア・システムの設計法」, インターフェース, No. 54, p. 197～217, 11 月, CQ 出版 (1981).

7) 相馬孝志：「68000 周辺回路, メモリ回路, および入出力回路の設計」, インターフェース, No. 54, p. 159～188, 12 月, CQ 出版 (1981).

8) Motorola Semiconductor Products Inc.：“*Motorola Microprocessors Data Manual*” (1981).

9) Motorola Semiconductor Products Inc.：“*MC* 68000 *FAMILY*”.

3章　68000のソフトウェア

本章では，68000のソフトウェアの概略をオペレーティング・システムとプログラミング言語の面から説明する．

オペレーティング・システムについては，ソフトウェア開発に関連してCP/M-68000とUNIXを，応用面に関連して68000RMSを説明する．

プログラミング言語についてはアセンブリ言語に重点をおいて説明する．ほかに，高級プログラミング言語についてはS-PL/HとFORTRANを簡単に述べる．

3.1　CP/M-68000

3.1.1　CP/M-68000の概要

（1）概　要

CP/M-68000 (CP/M-68K) は，マイクロコンピュータの最も標準的なオペレーティング・システムとして，世界中で広く使われているCP/M†の68000版で，Digital Research 社に(株)日立製作所が協力する形で開発が進められている．

CP/M-68Kの開発により，FORTRAN，Pascal，BASICなどの高級言語が手軽に利用できるようになり，これまでに高級言語を使って作成された80系および8086系の応用プログラムを，68000でも使えるようになる．CP/Mの下で動作する応用プログラムは数百種にも及ぶといわれており，それが68000でも利用可能となるわけで，その意味でもCP/M-68Kの開発の意義は大きい．

---

† CP/MはDigital Research社の登録商標である．

CP/M-68Kには次のような特徴がある.

i) 移植の容易さ

CP/M-68Kは, 構成が異なる各種のハードウェアで動作できるよう, ディスク, コンソール, プリンタなどの物理的構成からは独立した設計となっている. すなわち, 入出力装置に依存した部分は, 入出力装置に依存しない部分と分離した形で設計がなされており, 入出力装置に依存した部分は簡単なインタフェースで接続できるようになっている. したがって, ユーザがこれらの入出力装置の仕様に合わせたドライバ・ルーチンを組み込むことにより, CP/M-68Kは動作できるようになり, ユーザごとに構成の異なる各種のハードウェア・システムへの移植も容易に行なうことができる.

ii) 効率の良い操作性および資源の有効活用

CP/M-68Kは, コンピュータの操作を効率的に行なえるよう, また, ハードウェアの資源を有効に活用できるよう, 次の機能を備えている.

- ●プログラムの作成と修正
- ●プログラムの翻訳
- ●プログラムのロードおよび実行
- ●プログラムのデバッグ
- ●ファイルの作成
- ●ファイルの複写
- ●ファイルの消去
- ●ファイルの内容の表示
- ●ディスク・スペースの管理
- ●ディスクのディレクトリの表示
- ●ディスクおよび周辺装置の入出力機能のサポート

iii) 今後の開発予定

CP/M-68Kに続いて, マルチタスク機能をサポートするconcurrent CP/M, マルチ・ユーザ機能をサポートするMP/M-68000, ネットワーク機能をサポートするCP/NET-68000の開発が予定されている. いずれはOA (office automation), PWB (programmer's workbench) などのより高度な機能を実現するUNIXのサポートも実現されよう.

（2）　ハードウェア構成

CP/M-68Kで必要とする最小ハードウェア構成は，次の通りである．

- ●68000プロセッサ
- ●メモリ128Kバイト
- ●フロッピー・ディスクまたはハード・ディスク
- ●コンソール

このほか，プリンタ，紙テープ・パンチャ/リーダ，カセットテープなども接続することができる．

（3）　ソフトウェア構成

CP/M-68Kのソフトウェアの構成を，図3.1に示す．

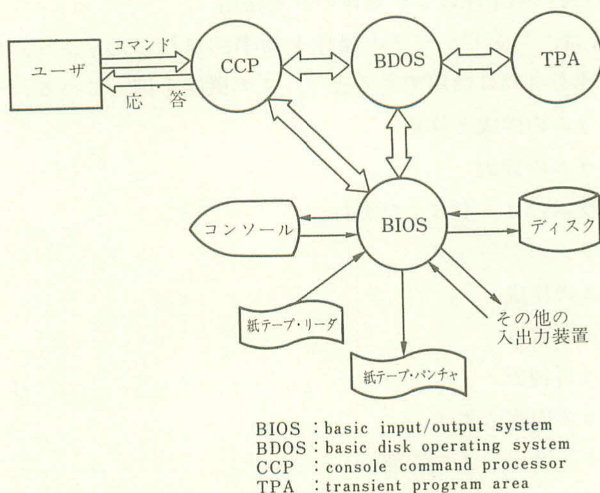


図 3.1　CP/M-68000のソフトウェアの構成

BIOS (basic input/output system) は，ディスク，コンソール，プリンタなどの入出力を制御するドライバ・ルーチンの集りであり，入出力機器とのインタフェースに必要な最も基本的な処理を行なうプログラムである．接続する入出力装置を変更する場合は，新しく接続する入出力装置の仕様に合わせてBIOSを変更しなければならない．

BDOS (basic disc operating system) は，入出力機器との間でデータの入出力を行なうプログラムである．多くのプログラムは，BDOSの入出力機能

を用いてデータの入出力を行なう．BDOSの入出力機能は，ディスクに対する入出力と，ディスク以外の入出力機器に対する入出力に2分される．

CCP (console command processor) は，コンソールから入力される各種のコマンドを解読してコマンドの内容に応じた処理を行なうプログラムである．頻繁に使用されるコマンドに対応する処理ルーチンは，メイン・メモリに常駐して，コマンドの入力と同時にただちに処理を行なうように構成されている．

TPA (transient program area) は，CCPによってディスクからロードされたプログラムを格納するエリアで，コマンドで指定されたプログラムはこの領域に移され，実行される．

3.1.2 コマンドとファイル

CP/M-68Kが動作可能となり，コマンド入力待ちになると

A>

がコンソールに表示される．Aはアクセスしているディスクのドライブ名（装置名）で，ドライブ名としてはAのほかにB，C，…，Pが使われる．

CP/M-68Kの標準コマンドを，表3.1に示す．コマンドはそのタイプにより，ビルトイン・コマンドとトランジェント・コマンドの二つに分けられる．ビルトイン・コマンドがCCPのメモリ領域に常駐して処理を行なうのに対し，トランジェント・コマンドはディスクからTPAにロードされて実行されるコマンドであることが二つのコマンドの大きな違いである．表3.1に示すトランジェント・コマンドのほかに，ユーザが作成したプログラムをトランジェント・コマンドとして登録，実行することもできる．

A>DIR (CR) （(CR): carriage return code)

で，ドライブ名Aのディスク上の全ファイル名が表示される．

ファイル名は

<名前>．<型名>

で示され，．<型名>はファイルの種類を示す（表3.2）．たとえば，FILE1という名前のアセンブラのソース・プログラムが格納されているファイルは

FILE1.ASM

となり，FILE2という名前のオブジェクト・プログラムが格納されているファイルは

FILE2.68K

となる．また，ファイル名とともにディスクのドライブ名を指定することがで

表 3.1　CP/M-68000 コマンド

	コマンド	機　　能	フォーマット
ビルトイン・コマンド	ERA	ファイル名で指定されたファイルをディスクから消去する	ERA␣ファイル名
	DIR	ファイル名で指定されたファイルのディレクトリを表示する．ファイル名を指定しない場合は，アクセスしているディスクの全ディレクトリの内容を表示する	DIR␣[ファイル名] （ファイル名は省略可）
	REN	旧ファイル名で指定されたファイルの名前を新ファイル名に変更する	REN␣新ファイル名＝旧ファイル名
	USER	Pを指定しなかったとき，現在のユーザ・エリアを表示し，Pがあればユーザ・エリアをユーザ・エリアPへ変更する（ユーザ番号の変更）	USER[P]
	TYPE	ファイル名で指定されたファイルの内容をコンソールに表示する	TYPE␣ファイル名
トランジェント・コマンド	STAT	現在割り当てられている物理装置名を表示する	STAT␣DEV：
		割当て可能な装置名を表示する	STAT␣VAL：
		指定された論理装置と物理装置の対応付けを行なう	STAT␣論理装置：＝物理装置
		指定されたドライブを読出し専用とする	STAT␣ドライブ名：＝R/O
		ファイルの状態（レコード・サイズ，エクステントなど）を表示する	STAT␣ドライブ名：ファイル名
		ファイルの大きさに関する情報を表示する	STAT␣ドライブ名：ファイル名$S
		ファイルの属性を読出し専用とする	STAT␣ドライブ名：ファイル名$R/O
		ファイルの属性を読書き兼用とする	STAT␣ドライブ名：ファイル名$R/W
		ファイルの属性をシステムとする	STAT␣ドライブ名：ファイル名$SYS
		ファイルの属性をディレクトリとする	STAT␣ドライブ名：ファイル名$DIR
	LO68	アセンブラや高級言語の出力したHEXファイルを機械語の実行可能な68Kファイルに変換する	LO 68␣{オプション名}␣ファイル名␣{<メッセージファイル名>}
	ED	指定したファイル名のファイルを作成または編集する．ファイルの作成や編集はサブコマンド（表3.3）を使って行なう	ED␣ファイル名
	DDT	指定したファイル名をもつプログラムをTPAにロードし，デバッグを行なう．デバッグはサブコマンド（表3.4）を使って行なう．	DDT␣ファイル名

（表 3.1の続き）

	コマンド	機　能	フォーマット
トランジェント・コマンド	AS68	ソースファイルをアセンブルしてリロケータブル・オブジェクトを生成する	AS68␣{オプション名}ソースファイル名 {リスティングファイル名}
	SUBMIT	ファイル名で指定されたファイル内にあるコマンド群を実行する．このとき，ファイル内の仮パラメータ $1，$2，…がパラメータ V1，V2，…で置き換えられる	SUBMIT␣ファイル名〔V1，V2，…〕
	PIP	次の機能がある ・ファイルのコピー ・入出力装置とディスク間のデータ転送 ・複数のファイルの転送	PIP␣ドライブ名：〔ファイル名〕 ＝〔ドライブ名：〕 ファイル名 （〔ファイル名〕，〔ドライブ名：〕は省略可） （注）ドライブ名としてLST，PUNなども指定可能

表 3.2 ファイルの型名（例）

型　名	意　味	型　名	意　味
ASM	アセンブラ・ソース	C	C言語ソース
HEX	絶対アセンブラ出力	CRL	Cコンパイラ・オブジェクト
PRN	リスト	MAC	相対アセンブラ・ソース
68K	実行形式	REL	相対アセンブラ・オブジェクト
BAK	エディタ・バックアップ用	CRF	クロス・レファレンス付きリスト
SUB	SUBMIT ファイル	FOR	FORTRAN ソース
BAS	BASIC ソース	COB	COBOL ソース
INT	インタプリタ・コード		

きる．ドライブ名がBの場合，上記のファイルは

B：FILE1.ASM
B：FILE2.68K

となる．たとえば

A>B：FILE2 (CR)

で，ドライブB上のFILE2.68Kファイルにあるプログラムが実行される．

ファイル名の指定では，＜名前＞と＜型名＞にワイルド・カード（wild card）を使用することができる．これを使えば

FILE1.＊はFILE1の名前をもつすべてのファイル

＊.ASMはASMの型名をもつすべてのファイル

を示す（“＊”がワイルド・カードである）．したがって，ASMの型名をもつす

べてのファイル名の表示を指示するコマンドは

```
A>DIR  *.ASM (CR)
```

となる．また

```
A>PIP  A:=B:FILE*.* (CR)
```

というコマンドは，ドライブBに格納されているFILEで始まる名前のすべてのファイルを，ドライブAに転送するコマンドで

```
A>PIP  A:=B:FILE1.ASM (CR)
A>PIP  A:=B:FILE2.68K (CR)
   ……
```

の一連のコマンドと同じ意味になる．

次に，プログラム（アセンブラ）を作成する場合を考えてみる．まずエディタでアセンブラによるソース・プログラムのファイルを作成する．作成に当たってはサブコマンド（表3.3）を使用する．

```
A>ED  PROG1.ASM (CR)   ……エディタの起動(PROG1. ASMはファイル名)
NEW FILE               ……PROG1.ASMファイルの生成(新しいときこう表示する)
*I (CR)                ……プログラムの挿入
………  }
………  } プログラム
∧Z                     ……挿入終了（∧Zはコントロール Z）
*E (CR)                ……エディタの終了，ファイルへ書き込む
```

次に，アセンブルは

```
A>ASM  PROG1 (CR)
```

で指示する．これにより，PROG1.ASMファイルのアセンブルが行なわれ，結果はA:PROG1.HEXファイルに出力される．アセンブルの結果をデバッ

表 3.3　EDのサブコマンド

サブコマンド	機　　能	サブコマンド	機　　能
B	カーソルを先頭に位置させる	I_S∧Z	Sを挿入
−B	カーソルを最後に位置させる	F_S	Sを捜す
±*n*L	カーソルを*n*行移動する	S_{S1}∧Z_{S2}	S1をS2に置き換える
±*n*C	カーソルを*n*文字移動する	V	行番号を付加する
±*n*T	*n*行表示	*n*:	カーソルを*n*行に位置させる
±*n*K	*n*行消去	E	エディタの終了
±*n*D	*n*文字消去		

∧Zはコントロール Zを意味する

表 3.4 DDTのサブコマンド

サブコマンド	機　　能
L　*n*, *m*	*n*番地から*m*番地のメモリの内容を記号命令で表示
D　*n*, *m*	*n*番地から*m*番地のメモリの内容を16進で表示
X	レジスタの内容を表示
X　*r*	レジスタ*r*の内容を変更
T　*n*	*n*命令をトレース実行
G　*n*, *m*	*n*番地から*m*番地まで実行
M　*a*, *b*, *c*	*a*番地から*b*番地の内容を*c*番地以降にコピー

グするには，DDT (dynamic debugging tool) を起動して，サブコマンド（表3.4）を使って行なう．

デバッグの終了したプログラムは

```
A>LOAD  PROG1 (CR)
```

のコマンドで機械語のPROG1.68Kファイルが作成され，コマンドで直接実行可能なプログラムとなる．

3.1.3 BDOSの機能

CP/M-68Kの入出力は，BDOS (basic disc operating system) を構成する各入出力サポート・ルーチンにより実行される．表3.5に主な入出力サポート・ルーチンを示す．これらの入出力サポート・ルーチンのコールは

```
MOVEQ.L  #<FC>, D0              機能コード (function code) をセット
MOVE.L  #<parameter>, D1        パラメータをセット
TRAP #2
```

で行なう．入出力サポート・ルーチンからのリターン情報は，レジスタD0にセットされている．

CP/M-68Kで扱う入出力装置は，ディスクを除くと，コンソール，リーダ，パンチャ，リストの4種類に限定されている．コンソールはキーボードとディスプレイ，リストはプリンタ，リーダとパンチャは紙テープが標準入出力装置として想定されている．これらの標準として想定された入出力装置に対し，実際に入出力を行なう装置はBIOSで変更可能であり，この意味で標準としての入出力装置は論理装置 (logical device) と考えることができる．論理装置と物理装置 (physical device) との対応は，IOバイト（IO BYTE）と呼ぶ1バイトのインディケータで行なう（表3.6）．

表 3.5 BDOSの入出力サポート・ルーチン

機能コード	意味	機能	パラメータ	リターン情報
0	システム・リセット	CP/M-68Kシステム(CCP)へ制御を渡す．CCPは，ディスク・ドライブAの選択およびディスク・システムのイニシャライズを行なう		
1	コンソール入力	コンソールからの入力をレジスタD0に格納する		ASCII文字
2	コンソール出力	ASCII文字をコンソールへ出力する	ASCII文字	
3	リーダ入力	リーダ（論理装置）からの入力をレジスタD0に格納する		1バイト・データ
4	パンチャ出力	パンチャ（論理装置）の出力	1バイト・データ	
5	リスト出力	リスト（論理装置）の出力	ASCII文字	
6				
7	I/Oバイト取出し	IOバイトの値をレジスタD0に格納する		I/Oバイト
8	I/Oバイト変更	IOバイトの値を指定された値に変更する	I/Oバイト	
9	コンソールへの連続文字出力	出力バッファに格納されている文字列のコンソールへの出力．文字列の最後を示す“$”まで出力する	出力バッファのアドレス	
10	コンソールからの連続文字入力	コンソールから入力された文字列を入力バッファに格納する	入力バッファのアドレス	ASCII文字列
11	コンソールの入力チェック	コンソールの入力データの有無のチェック		入力なしのとき0
12				
13				
14	ディスク選択	指定された番号のディスクを以後のファイル操作の標準ディスクとする	ディスク・ドライブ番号	
15	ファイル・オープン	指定されたファイルを利用可能状態とする	FCBの番地	ディレクトリ内のファイル番号
16	ファイル・クローズ	ディスクのディレクトリに最新のFCBの内容を格納する	FCBの番地	ディレクトリ内のファイル番号
17				
18				
19	ファイルの消去	FCBで示されたファイルを消去する	FCBの番地	
20	1レコード・リード	ファイルから128バイトのデータを読み込み，DMAアドレスで示されるメモリに格納する	FCBの番地	0：正常終了 1：ファイル終了 2：存在しない
21	1レコード・ライト	DMAアドレスで示される128バイトのデータをFCBへ示されるファイルへ書き込む	FCBの番地	0：正常終了 1：領域不足 255：ディレクトリ不足
22	ファイルの作成	ディレクトリとメモリのイニシャライズを行ない，データのないファイルを作成する	FCBの番地	ディレクトリ内の番号 255：ディレクトリ不足

(表3.5の続き)

機能コード	意味	機能	パラメータ	リターン情報
23	ファイル名の変更	ファイル名を新しいファイル名に変更する	FCBのアドレス 新しいファイル名(FCBのEX以降の8バイトにセット)	255：変更するファイルが存在しない
24				
25	選択されているディスク番号の取出し	現在選択されている標準ディスク・ドライブの番号を取り出す		ディスク・ドライブ番号
26	DMAアドレス・セット	DMAアドレスのセット	DMAアドレス	
27				
28				
29				
30				
31				
32				
33	ランダム・レコード・リード	FCBで指定されたレコード番号のレコードを読み込み，DMAアドレスで示されるメモリに格納する	FCBのアドレス	0：正常終了 1：ファイル終了 2：存在しない
34	ランダム・レコード・ライト	FCBで指定されたレコード番号のデータをファイルに書き込む．データはDMAアドレスで示されるメモリに格納されている	FCBのアドレス	0：正常終了 1：領域不足 255：ディレクトリ不足
35	レコード数の取出し	FCBの33～35バイトに次にエンド・オブ・ファイルとなるレコード番号をセットする	FCBのアドレス	
36				
37				
40				
46				
50	BIOSコール	ユーザ・プログラムから直接BIOSの機能ルーチンをコールする．パラメータ・ブロックの構成は次の通り BIOS機能コード（2バイト） パラメータ1 （4バイト） パラメータ2 （4バイト）	パラメータ・ブロックのアドレス	BIOSコールの結果
59	プログラム・ロード	パラメータ・ブロックで示されたプログラムを指定アドレスにロードする パラメータ・ブロックの構成は次の通り プログラム・ファイルのFCB ロード・トップ・アドレス ロード・エンド・アドレス	パラメータ・ブロックのアドレス	

表 3.6 IOバイト

7 6	5 4	3 2	1 0
list	punch	reader	console

	0	1	2	3
list （リスト）	TTY	CRT	LPT	UL1
punch （パンチャ）	TTY	PTP	UP1	UP2
reader （リーダ）	TTY	PTR	UR1	UR2
console （コンソール）	TTY	CRT	BAT	UC1

TTY：teletypewriter　　LPT：line printer
UL　：user list　　PTP：paper tape puncher
UP　：user puncher　　PTR　paper tape reader
UR　：user reader　　BAT：batch
UC　：user console

BDOSでは，以上の論理装置に対し，表3.5の機能コード（function code）0～11の入出力サポート・ルーチンが準備されている．

CP/M-68Kのファイルのアクセスは，FCB（file control block）を通して行なわれる．FCB（図3.2）は，36バイト（シーケンシャル・ファイルの場合は33バイト）で構成されており，32バイトのディレクトリに1～4バイトの

DR：ディスク・ドライブ・コード（0～16）
　0：標準ディスク・ドライブ（コマンドで選択したディスク）
　1, 2, 3, …, 16でA, B, C, …, Pのディスク・ドライブを指定する
FN：ファイルの名前（ASCII）
FT：ファイルの型名（ASCII）
EX：ファイルのエクステント（extent）番号（0～31）
SYS：（システム用）
RC：記憶しているレコード数
DM：ディスク・アロケーション・マップ
SRN：読出し，書込みを行なっているレコード番号（シーケンシャル・ファイル）
RRN：読出し，書込みを行なっているレコード番号（ランダム・ファイル）

図 3.2　FCBの構成

情報を付加した構成となっている．ディレクトリはディスクのディレクトリ領域に格納されており，ファイル操作の前にメモリにロードされる（ファイル・オープンおよびファイル作成時）．メモリにロードされたディレクトリは，ファイル操作が行なわれるにつれて内容が更新され，ファイルのクローズ要求でディスクのディレクトリ領域に格納される．

ファイルは，ディスク上では1Kバイトのブロック単位に確保され，メモリとのデータの転送は，128バイトのレコード単位で行なう．ディレクトリでは，各ファイルに対し16バイトのディスク・アロケーション・マップ（disc allocation map）を用意し，合計16Kバイトの領域を記憶することができる．16Kバイトで領域が不足するときは，ディレクトリの"EX"バイトを使って領域を

表 3.7 BIOSの機能ルーチン

機能コード	名　称	機　能
0	INIT	ローダから制御を受け取り，次の処理を行なった後でCCPに制御を渡す ・システム・ロードの終了を示すメッセージの出力 ・標準ディスクとしてディスク・ドライブAを選択 ・システム・パラメータのセット
1	WBOOT	システム・パラメータのイニシャライズ後，CCPへジャンプする
2	CONST	コンソールのキー・イン・データの有無に応じてレジスタD0にステータスをセットする
3	CONIN	コンソールからのキー・イン・データをレジスタD0に読み込む
4	CONOUT	レジスタD1にセットされているデータをコンソールへ出力する
5	LIST	レジスタD1にセットされているデータを現在割り当てられているリストへ出力する
6	PUNCH	レジスタD1にセットされているデータを現在割り当てられているパンチャへ出力する
7	READER	現在割り当てられているリーダからレジスタD0にデータを読み込む．エンド・オブ・ファイルはコントロール-Z ($1A) でリターンする
8	HOME	現在選択しているディスク・ドライブのヘッドをトラック0へ移動
9	SELDSK	レジスタD1にセットされている値に従ってディスク・ドライブを選択．D1=0, 1, 2, …, 15に従ってドライブA, B, C, …, Pを選択する
10	SETTRK	レジスタD1にセットされている値をトラック番号とする
11	SETSEC	レジスタD1にセットされている値をセクタ・アドレスとする
12	SETDMA	レジスタD1の値をディスク・データ転送時のDMAアドレスとする
13	READ	1セクタのデータを読み込む．エラーが発生したときレジスタD0に1，正常のときは0をセットしてリターンする
14	WRITE	1セクタのデータを書き込む．エラーが発生したときレジスタD0に1，正常のときは0をセットしてリターンする
16	SECTRAN	論理セクタ・アドレスを物理セクタ・アドレスに変換する
19	GETIOBYT	レジスタD0に現在のIO BYTEをセットしてリターンする
20	SETIOBYT	レジスタD1の値を新しいIO BYTEとする

拡張することができる．

3.1.4 BIOSの機能

CP/M-68Kをそれぞれのシステムに移植するには，システムの仕様に合わせてBIOS (basic I/O system) を変更しなければならない．

BIOSは，BDOSと同じく，種々の機能ルーチンから構成されており，これらの機能ルーチンのコールは

```
MOVEQ.L  #<FC>, D0           機能コード (function code) をセット
MOVE.L   #<parameter>, D1    パラメータをセット
TRAP     #3
```

で行なう．パラメータの受渡しが二つ以上の場合は，必要に応じてレジスタD2，D3，…を使う．表3.7に，BIOSの機能ルーチンの主なものを示す．

3.2 UNIX†

3.2.1 UNIXの概要

16ビット・マイクロコンピュータのオペレーティング・システムとして，UNIXが注目されている．UNIXは，アメリカのBell研究所が開発したコンパクトなTSSオペレーティング・システムである．開発当初はミニコンピュータPDP-11用に開発されたが，マイクロコンピュータの性能がかつてのミニコンピュータの性能を凌駕するに及んで，UNIXがマイクロコンピュータ用の高級オペレーティング・システムとして脚光を浴びるようになった．

（1） 代表的なUNIXシステム

Bell研究所のトンプソン (K. Thompson) によりUNIXが開発され始めたのは，1969年であった．最初はPDP-7用の単一ユーザむけシステムであったが，ついでPDP-11/20用のUNIXが開発され，1973年にはそれまでアセンブラで書かれていたUNIXの大部分が，一部を除いて高水準言語Cで書き直された．現在ではUNIXの種類もいろいろに分かれてはいるが，UNIXが使いやすい気の利いたシステムとして高い評価を受けていることには変わりはない．

表3.8に，代表的なUNIXシステムを示す．

---

† UNIXはBell研究所の登録商標である．

表 3.8 代表的なUNIXシステム[4)]

	16ビット用				32ビット用
UNIXのバージョン 発表年	MINI-UNIX	V6 1976	V7	PWB 1976	32V/4.1BSD 1981
最下位CPU 最小記憶 OS領域 最大端末数	PDP-11/10 56 KB 24 KB 4	PDP-11/34 96 KB 40 KB 40	PDP-11/45 256 KB 96 KB 40	PDP-11/45 192 KB 96 KB 48	VAX-11/750 512 KB

KB：キロ・バイト

68000で使えるUNIXシステムとしては，Microsoft社のXENIX（V7相当），Unisoft社のUNIX VⅢ（V7＋PWB相当），Cromemco社のCROMIX，Motorola社のSYSTEM V/68などがある．

（2） UNIXの特徴

以下にUNIXの特徴を示す．

i） UNIXのファイル・システムはきわめて簡単で，ファイル編成，レコード長，ブロック長，サイズなどの概念は一切必要ない．ファイルは単なる文字列の集りであり，ファイルのディレクトリ（directory）はトリー構造となっている．

ii） 入出力装置とファイルが全く同格に扱われる．端末のキーボードの入力の代わりにファイルから入力させたり，出力を端末のディスプレイやプリンタに出す代わりにファイルへ出したりすることが簡単にできる．

iii） コマンドはシェル（shell）と呼ばれるインタプリタ（interpreter）で解釈される．コマンドとしては，プログラム（メモリに格納されるバイナリ・イメージ）のファイル名や，いくつかのコマンドを入れたコマンド・ファイル名も許される．また，コマンド自身を一種のプログラム言語として扱うことができ，if … then … else … , while … do … done … などの構文も使用できる．したがって，いろいろなコマンドを組み合わせて新しいコマンドを作ることも簡単にできる．

iv） フィルタ（filter）と呼ぶコマンドを連続して実行させるとき，コマンド間のデータの受渡しは，パイプと呼ばれるファイルを経由して自動的に行なわれる．

```
command1 | command2 | command3
```

の"|"記号が，コマンドの間をつなぐファイルを表わす．"|"があると，コマンドの出力はパイプに入り，次のコマンドへの入力はパイプから入力され，ファイル名はいちいち指定する必要がない．

v）コマンドの同時並行処理の指定が可能である．

command1 & command2

と入力すると，command1の処理と並行して同時にcommand2の処理が行なわれる．

vi）UNIXの大部分が高水準言語Cで書いてあり（C約1万行，アセンブラ約1000行），保守や変更が容易である．

vii）いろいろなプログラミング言語やユーティリティ・プログラム（utility program）が用意されている．

- C，FORTRAN77，BASIC，ALGOL68，Pascal，APL，SNOBOL
- アセンブラ，リンクローダ，シンボリック・デバッガ
- テキスト・エディタ，清書プログラム

など

viii）使用文字は小文字がベースとなっている．

3.2.2　UNIXのファイル・システム

UNIXにおいてファイル・システムの果たす役割は重要である．ファイル・システムの構成はきわめてシンプルで，ユーザの側から見たとき，ファイルは，通常のファイル，ディレクトリ，特殊ファイルの3種類に分けて考えることができる．

通常のファイルでは，システムがファイルの中身に特別な構造を規定することは一切ない．ユーザは，必要に応じてどんな情報でもファイルに格納しておくことができる．たとえば，テキスト・ファイルは単なる文字列の集りで，行単位に改行文字で区切ったものであるし，バイナリ・プログラム（binary program）のファイルは，プログラム実行時のメイン・メモリ上のプログラムをそのままファイルに並べただけのものである．オブジェクト・プログラム（object program）のように，特別なフォーマットを必要とするものもあるが，このようなファイルは，コンパイラやアセンブラが特別なフォーマットのファイルも作り出し，ローダ（loader）がその特別のフォーマットのファイルを受け付けるようになっている．すなわち，ファイルの構造を制御するのはあくまでもプロ

グラムであり，システムではないということである．

ディレクトリ (directory) は，ファイルの名前（長さ14文字以下の文字列）とファイルの実体とを対応付けるためのものであり，その構造はツリー構造 (rooted tree) となっている（図3.3）．通常のファイルおよび特殊ファイルはすべてこのツリー構造の末端につながれる．UNIX では，ユーザのファイル全体のためのディレクトリがユーザごとに一つずつあり，さらにユーザがそのディレクトリの中に下位のディレクトリを作って，その下でファイルを一括して扱うこともできる．ディレクトリは通常のファイルと全く同様に処理されるが，ディレクトリの内容（ファイル名とポインタ）はシステムだけが変更できるようになっている．しかし，許可が与えられれば，ユーザもディレクトリの内容をファイルと同様に読むことができる．

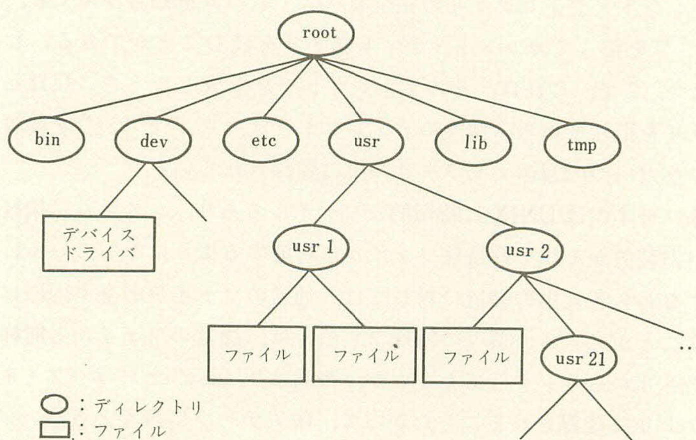


図 3.3 ディレクトリのツリー構造

システムに対するファイル名の指定は，パス・ネーム (path name) により行なうのが一般的である．パス・ネームとは，“／”で区切ってディレクトリ名を並べ，最後がファイル名で終わるような形の文字列である．パス・ネームが“／”で始まっているときは，ルート・ディレクトリ (root directory) の中から探し始める．たとえば

／alpha／beta／gamma

というパス・ネームならば，ルート・ディレクトリの中で alpha というディレ

クトリを探し，次のそのalphaの中からbetaを探し，最後にbetaの中からgammaというファイルを探す．gammaは通常のファイルでも特殊ファイルでもよく，どちらでもかまわない．“／”はディレクトリのツリー構造の階層を示す区切り記号であるが，“／”単独で用いられたときは，特にルート・ディレクトリ自身を示している．“／”で始まっていないパス・ネームが指定されたときは，ユーザのカレント・ディレクトリ（current directory）の中を探す．たとえば

beta／gamma

というパス・ネームは，カレント・ディレクトリの下位のディレクトリであるbetaの中のgammaというファイルを示している．

各ディレクトリには，少なくとも二つの名前が必ず登録されている．そのうちの一つは“．”という名前で，そのディレクトリ自身を指している．したがって，カレント・ディレクトリの完全なパス・ネームを知らなくても，プログラムは“．”を使ってカレント・ディレクトリを読むことができる．もう一つの名前は“．．”で，これは，そのディレクトリを作成したときに登録した親ディレクトリを指している．個々のディレクトリは，“．”と“．．”を除けば，自分以外のただ一つの親ディレクトリの中に登録されている．

特殊ファイルはUNIXに特徴的なファイルであり，システムに接続されている入出力装置がそれぞれ特殊ファイルに対応するようになっている．特殊ファイルすなわち各入出力装置に対しては，通常のファイルと全く同様に入出力を行なうことができる．個々の特殊ファイルは，通常のファイルと同様にどのディレクトリからリンクしてもよいが，普通は“／dev”というパス・ネームのディレクトリに登録される．したがって，磁気テープに書くときは，／dev／mtというファイルに書けばよいことになる．特殊ファイルは，通信回線，ディスク，磁気テープおよびメイン・メモリに対応するものが存在する．

図3.4に，UNIXのファイル・システムの内部構成を示す．ディレクトリのリンク（ファイルを登録する場所）には，ファイルの名前とファイルのポインタの二つが書かれている．ポインタはi-number（index number）と呼ばれる整数で，システム・テーブルi-listのインデックスとして使われる．i-listはディレクトリが存在するのと同じディスク上の決められた場所に格納されている．i-listのそれぞれの要素はファイルのi-nodeと呼ばれ，ファイルの所有者のユーザ番号とグループ番号，ファイルの保護ビット，ファイルの実体が格納され

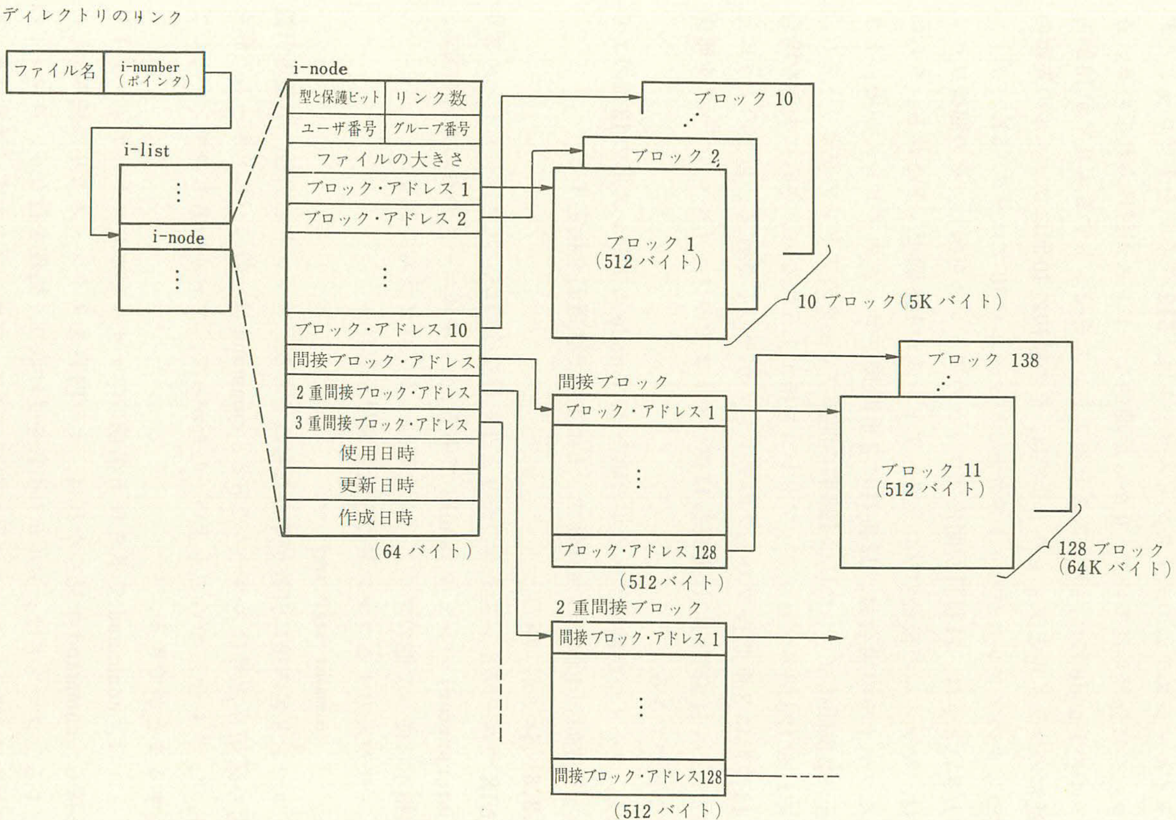


図 3.4 ファイル・システムの内部構成

ているディスク上のアドレスなどの情報がセットされている．

ファイル・システムのディスク・スペースは，512バイトを1ブロックとするブロックに分けられていて，0から論理的なアドレスが付けられている．各ファイルのi-nodeには，ディスク上のアドレスをセットするエリアが13個あり，通常のファイルとディレクトリでは，その最初の10個にファイルの先頭の10ブロックのアドレスがセットされる．ファイルが10ブロック（5Kバイト）より大きいときは，11番目が間接ブロックのアドレスを指し，その間接ブロックには，ファイルの後続のブロック・アドレスが128個セットされる．さらに大きなファイルの場合は，12番目に2重間接ブロックのアドレスがセットされ，その2重間接ブロックには128個の間接ブロックのアドレスがセットされ，それぞれが128個のブロック・アドレスを指すようになる．さらに大きな場合は，13番目に3重間接ブロックのアドレスがセットされる．以上のようにして，ファイルは論理的には最高約11億バイトまでのファイル・スペースを確保することができる．

一方，ファイルが特殊ファイルの場合は，i-nodeのアドレス用の13個のエリアのうち先頭の1個のみが有効で，その1個で装置名を指定する．

3.2.3　シェル

UNIXでユーザとシステムのやりとりを行なうのが，コマンド・インタプリタ (interpreter) のシェル (shell) である．シェルはユーザからの入力を読み，その指示に従って他のプログラムを起動するプログラムである．

ユーザが入力するコマンド行の最も簡単な形は

```
command  arg1  arg2  ……
```

で，コマンド名の後に引数が並んだものである．コマンドと引数，引数と引数の間の区切り記号はスペースである．commandは一般にパス・ネームの形をとり，“／”を含んでいてもよい．したがって，ファイル名もコマンドとして指定することができる．

シェルは，commandで示される名前のファイルをカレント・ディレクトリから探し，commandが見つかれば，その内容をメイン・メモリに読み出して実行する．コマンドはシェルから引数を受け取り，処理を行なう．コマンドの処理が終了すると，シェルに再び制御が戻り，プロンプト文字（通常は＄）を出力して次のコマンドの入力を待つ．commandというファイルが見つからなか

った場合，シェルはあらかじめ指定された順序で他のディレクトリ（たとえば，一般ユーザ用のコマンドを登録しておくディレクトリなど）を探す．

シェルによって起動されたプログラムは，ファイル番号0, 1, 2で示される三つのファイルがオープンされた状態で実行を開始する．ファイル番号0が標準入力，ファイル番号1，2が標準出力で，標準入力は端末のキーボードに，標準出力は端末のディスプレイ（またはプリンタ）に割り当てられている．

シェルはファイル番号0, 1で示されるファイルを，“<”や“>”を使って端末のキーボードやディスプレイ以外のファイルに割り当てることもできる．たとえば

```
ls
```

というコマンドでは，カレント・ディレクトリの中のファイル名のリストをディスプレイに出力する．これに対して

```
ls  >listfile
```

というコマンドでは，listfile というファイルを作り，リスティングをそのファイルに対して行なう．すなわち，>listfile という引数は“出力をファイル listfile に対して行なえ”という意味になる．また

```
ed  <sourcefile
```

というコマンドの引数 <sourcefile は，キーボードからではなく“ファイル sourcefile から入力せよ”という意味になる．

ファイル番号2は，ファイル番号1と同じく標準出力であるが，エラー・メッセージの出力に使うので，他のファイルに切り替えられることはなく，端末のディスプレイに割り当てられたままである．

表3.9に，入出力の切替えを示す．

標準入出力の拡張としてパイプラインの機構がある．これは“|”で区切ったコマンド列が与えられた場合，“|”の左側にあるコマンドの標準出力が，パイ

表 3.9 入出力の切替え

引　数	標準入力	標準出力
指定なし	キーボード	ディスプレイ
>	キーボード	ファイル
<	ファイル	ディスプレイ
< >	ファイル	ファイル

プ (pipe) と呼ぶファイルを通して"|"の右側にあるコマンドの標準入力となる機構のことである．パイプと呼ぶファイルの作成，消去はシステムで自動的に行なわれ，ユーザが意識する必要は全くない．たとえば

```
ls  |  pr −2  |  opr
                  └─入力をオフライン出力用ファイルにスプールする
        └─ページ分けして，タイトルを付けて印刷する
          (−2の引数は2列に分けて出力することを意味する)
 └─カレント・ディレクトリのファイル名のリストを作る
```

というコマンド列は，"|"を使わずに書くと次のようになる．

```
ls   >temp1
pr   −2  <temp1  >temp2
spr  <temp2
```

prのように，パイプラインを表わす"|"の次に書けるコマンドは，フィルタ (filter) と呼ばれる．

シェルでは，コマンドの連続処理と並列処理が可能である．複数のコマンドを";"で区切って同じ行に入力すると，それらのコマンドは出現順に連続して実行される．たとえば

```
ls ; ed
```

というコマンド入力では，まずカレント・ディレクトリの内容を出力し，次にエディタの処理が行なわれる．また，コマンドの後に"&"を付けて入力すると，シェルはコマンドの終了を待たずにすぐにプロンプト文字を出力して，次のコマンドの入力待ち状態になる．たとえば

```
as  source  >output  &
```

が入力されると，sourceのアセンブルを開始するが，このアセンブル処理とは無関係にシェルはすぐにコマンド入力待ち状態となる．"&"は1行の中で2回以上使うことも可能で

```
command1  &  command2  &  command3
```

と入力すると，同時に三つのコマンドが実行される．このほか，(　)を使うことも可能で，たとえば

```
(date ; ls) >x  &
```

と入力すると，最初に日付と時刻，次にカレント・ディレクトリのリストがファイル x に書かれる．この場合もコマンドの終了を待たずにシェルはすぐにコ

表 3.10 UNIXの代表的コマンド例[4]

	コマンド	機能	備考
一般コマンド	date	日付，時刻を表示	
	echo string	文字列をそのまま表示	
	login username	システムの使用開始を宣言	
	ls-1	ディレクトリのファイル名をリスティングして表示	
	man	マニュアルを表示	
	pwd	カレント・ディレクトリのパス・ネームを表示	
	sleep 5	5秒間処理を休止	
	time command	指定されたコマンドの実行時間を測る	
	who	現在使用中のTSSユーザのリストを表示	
ファイル関連コマンド	cat file	ファイルの内容を表示	フィルタ
	cmp file1 file2	ファイルを比較し異なる部分を出力	
	comm file1 file2	ファイルを比較し一致する部分を出力	フィルタ
	cp file1 file2	ファイル1をファイル2にコピー	
	ed file	指定されたファイルのエディション	フィルタ
	grep expr file	ファイルの中から文字列を探す	フィルタ
	mv file1 file2	ファイル名を変更	
	pr −2 file	ファイルを2カラムにプリント	フィルタ
	rm file	指定されたファイルを削除	
	sh file	指定されたファイル名のコマンド・ファイルを実行	フィルタ
	sort file	指定されたファイルの内容をソート	フィルタ
	split −3 file	ファイルを3行ずつに分割	
	sum file	指定されたファイルの内容のチェック・サム	
	unig file	隣り同士同じ行は1行にまとめる	フィルタ
	wc file	指定されたファイルの行数，語数，字数をカウント	フィルタ

マンド入力待ち状態となる．

表3.10に，UNIXの代表的なコマンドの例を示す．

3.3 68000 RMS

3.3.1 68000 RMSの概要

68000 RMS (real time monitor system) は，プロセス制御を中心とする広範な分野の応用システムにおいて，リアルタイム処理およびマルチ・プログラミング (multiprogramming) 機能を容易に実現することを目的としたスーパバイザ・プログラム (supervisor program) である．RMSは，HD68000シングル・ボード・コンピュータ (H680SB01) およびモニタ・ボード (H680MN01)

で使用されるファームウェア（EPROM　HN462732×6個）として（株）日立製作所から提供されている．

（1）　RMSの機能

RMSの主な機能には，次のようなものがある．

- タスク（RMSが管理するユーザ・プログラムの単位）に付けられた優先順位に従って，タスクの起動スケジューリングを行なう．
- RMSは，プロセスの変化に伴って発生する割込み信号や入出力データの転送の待ち時間を利用して，複数のタスクの同時処理を行なう．
- タスク制御マクロ命令によって，タスクの動作を自由に変更することができる．
- タスク間の同期（synchronization）と連絡を行なうことができる．
- タスク間でメッセージの送信/受信が可能である．
- ACIA（HD46850 asynchronous communication interface adapter）およびプリンタの入出力制御を行なう．ユーザ独自の入出力装置については，ユーザの作成したドライバ・ルーチンをRMSに組み込むことにより簡単に制御できる．このACIAはデータフォーマッティングや制御を行なうLSIで，そのコントロールレジスタはプログラム可能である．
- ユーザは，割込み分析，入出力制御，マクロ命令，イニシャライズ用の各プログラムを作成し，RMSに組み込むことができる．
- ソフトウェア・タイマが利用できる．
- オンライン・デバッグが可能である．

（2）　RMSのプログラム構成

図3.5に，RMSのプログラム構成を示す．

i）　外部割込み処理

入出力装置やタイマなどの割込みを分析して，それぞれの割込みに対応したプログラムの処理を実行する．RMSが備えている割込み処理機能以外の処理が必要であれば，必要な割込み処理プログラムを作成してRMSに組み込めばよい．これをRMSの一部として実行することができる．

ii）　エクセプション割込み処理

バス・エラー，アドレス・エラー，ゼロによる割り算などによって発生するエクセプションに対し，エクセプション割込み処理を実行する．

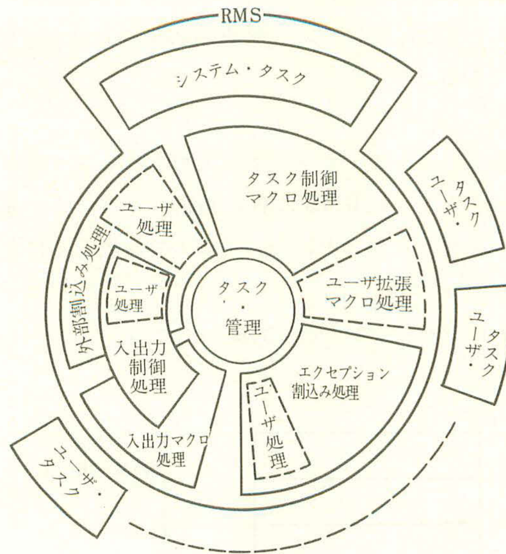


図 3.5　RMSのプログラム構成

iii）　タスク制御マクロ処理

タスクから発行されるタスク制御マクロ命令の処理を行なう.

iv）　入出力マクロ処理

タスクから発行される入出力マクロ命令の処理を行ない，入出力を要求された装置の入出力制御プログラム（ドライバ・ルーチン）に制御を渡す.

v）　入出力制御処理

入出力データの転送や入出力に伴って発生するエラーの処理を行なう. RMSでは，ターミナル・コンソールとプリンタ用の入出力制御プログラムを標準としてもっており，他の入出力装置に対しては，対応する入出力制御プログラムを作成して RMS に組み込むことにより，RMS の入出力マクロ命令で入出力処理を行なうことができる.

vi）　タスク管理

タスクの状態を管理じ，タスクの優先順位に従ってどのタスクを実行させるかを決定し，タスクを起動する.

vii)　ユーザ拡張マクロ処理

RMSのマクロ命令以外に，ユーザ専用のマクロ命令を作成してRMSに組み込むことができる．

viii)　システム・タスク

システム・タスクは，コンソールから入力されるコマンドの処理を行なう．

i)，v)，viii)のようにRMSに各種制御プログラムを組み込めるが，その際どんな割込みが使えるかなどについては十分注意してほしい．

(3)　メモリ・マップ

RMSのメモリ・マップの例を図3.6に示す．

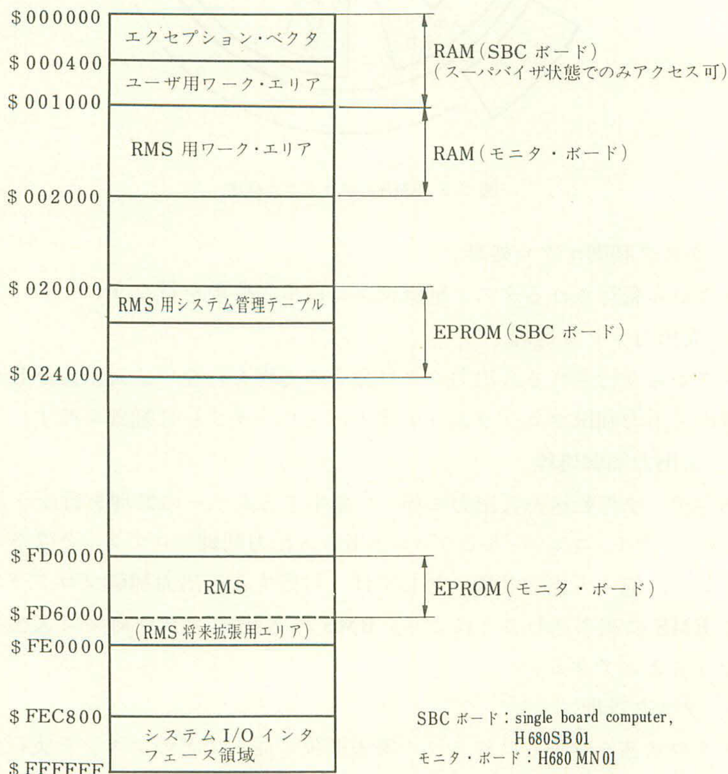


図3.6　RMSメモリ・マップ例

3.3.2 RMSの機能と構造

(1) 割込みとRMSの構造

i) 割込み

割込みとは，MPUの処理とは無関係に非同期に発生するある要因によって，それまで実行中の処理を中断してその要因に対応するプログラムに制御を渡す機能で，RMSでは割込みを次のような目的で使っている．

①IRQ (interrupt request)

IRQ割込みには，レベル1からレベル7までの割込みがある．レベル1～6の割込みには，入出力割込み，プロセス割込み，タイマ割込みなどがあり，ステータス・レジスタの割込みマスク・ビットで割込みをマスクすることができる．これに対し，レベル7の割込みはノン・マスカブル (non-maskable) な割込みとして利用することができる．

②TRAP

TRAP命令による割込みである．TRAP割込みにはTRAP 0からTRAP 15までがあり，TRAP 0，TRAP 1はRMSのマクロ命令コールに利用されている．TRAP 2～15はユーザ用で，タスク内のソフトウェア割込み，あるいは，TRAP処理タスクの処理依頼に使われる．

③RESET

リセット割込みで，RMSはパワーオン時のシステム起動を行なう．また，RMSのイニシャライズにも利用できる．

④その他の割込み

上記以外にも，バス・エラー，アドレス・エラー，不当命令，ゼロによる割り算，CHK命令，TRAPV命令，特権違反，ライン1010および1111エミュレータの割込みがある．RMSではこれらの割込みに対し同一処理を行なっている．これらの割込みは，システムの異常やタスクの異常検出に利用することができる．

なお，スプリアス (spurious) およびトレースの割込みはRMSで使用しているので，ユーザは使用できない．

ii) RMSの割込み処理

①IRQ割込み

a) オート・ベクタ割込み（ベクタ番号25～31）

オート・ベクタ割込みが発生すると，RMSはシステム・テーブルで定義さ

れた割込み要因テーブルに従って割込み要因を調べ，それぞれの要因に対応する処理を行なう．

- タイマ割込み：SBCボード上のプログラマブル・タイマ（HD46840 PTM）からの割込みで，時刻の更新，時間経過によるタスクの起動などの処理を行なう．
- 入出力装置からの割込み：入出力転送の終了処理，次のデータの転送処理，タスクの起動処理などを行なう．
- プロセス割込み：PIA（HD46821 peripheral interface adaptor）からの割込みに対し，割込み要因に対応するタスクの起動処理やイベントの発生処理を行なう．このPIAは，2個のコントロールレジスタをもつLSIである．
- その他の割込み：割込み要因テーブルで指定されているユーザ作成の割込み処理プログラムに制御を渡す．

b）ユーザ・ベクタ割込み

オート・ベクタ以外の割込みは，RMSを介さずに割込みの発生から直接ユーザ・プログラムへ制御が渡される．

②TRAP

a）TRAP 0

TRAP 0は，スーパバイザ状態でのRMSに対する処理要求に使われる．すなわち，RMSに組み込まれ，RMSの一部としてスーパバイザ状態で実行されるユーザ・プログラムからのマクロ命令コールに用いられる．

b）TRAP 1

TRAP 1は，ユーザ・タスクからのマクロ命令コールとして用いられる．

c）TRAP 2～15

TRAP 2～15がユーザ・タスクから発行されると，RMSは次の処理を行なう．

- TRAP 2～15を発行したタスクが，そのTRAPに対応する処理プログラムを指定（TRPVCTマクロで指定可能）している場合は，対応するTRAP処理プログラムに制御を渡す．
- 対応するTRAPを処理するサーバ・タスク（server task）が定義（SERVEマクロで定義可能）されている場合は，TRAP発行タスクの処理を中断し，対応するサーバ・タスクにTRAP発生のメッセージを

送り，TRAP が発行されたことを連絡する．TRAP を発行したユーザ・タスクは待ち状態となる．

●上記以外の場合は，TRAP を発行したユーザ・タスクにエラー・リターンする．

③RESET

RMS のイニシャル処理を行なう．システム・テーブルにユーザが作成した RESET 処理ルーチンが指定されている場合には，その処理ルーチンの処理も行なう．

④その他の割込み

バス・エラー，アドレス・エラー，不当命令，ゼロによる割り算，CHK 命令，TRAPV 命令，特権違反，ライン 1010 および 1111 エミュレータの各割込みが発生した場合には，RMS は次の処理を行なう．

a） ユーザ・タスク実行中に上記割込みが発生した場合

●割込みの発生したタスクが，その割込みに対応する処理プログラムを指定している場合には，対応する処理プログラムに制御を渡す．

●割込みの発生したタスクが，その割込みに対応する処理プログラムを指定していない場合には，割込みの発生したタスクの実行を打ち切り，実行禁止状態とする．

b） RMS 実行中に上記割込みが発生した場合

RMS，またはユーザが RMS に組み込んだ（サブ）ルーチンにおいて上記割込みが発生した場合，オンライン処理続行不可能とみなし，システム・テーブルで指定されたプログラムに制御を渡す．

（2） RMS とタスクのリンケージ

RMS とタスクのリンケージは，RMS で用意しているマクロ命令 CRTASK を利用して行なわれる（p. 159 参照）．

i） マクロ命令の処理

タスクからマクロ命令がコールされると，RMS は各種のレジスタの内容をタスクのレジスタ退避エリアに退避し，そして，タスクから渡される情報を取り込んでマクロ命令の処理を行なう．マクロ命令の処理が終わると退避中のレジスタを回復し，マクロ・コール命令の次の命令からタスクの処理を続行する．

なお，マクロ命令のアドレスをシステム・エディション時に RMS に与える

ことにより，RMSにないユーザ専用のマクロ命令をRMSに組み込むことも可能である．

ii）マクロ命令の一般的規則

①マクロ・コード

マクロ命令の種類を示すパラメータで，データ・レジスタD0にセットする．

②パラメータの渡し方

パラメータが1個の場合は，アドレス・レジスタA0またはデータ・レジスタD1にパラメータをセットし，パラメータが2個以上の場合は，アドレス・レジスタA0にパラメータの格納アドレスをセットしてマクロ命令をコールする．

③リターン・コード

RMSでは，データ・レジスタD0にマクロ命令の処理結果を示すコードをセットして，タスクにリターンする．この場合，コンディション・コード・レジスタのゼロ・ビット(Z)には，データ・レジスタD0にセットされているリターン・コードの値が反映されている．

```
         MOVE.L  #PARAM.A0    パラメータの格納アドレスをセット
         MOVE.L  #n, D0       マクロ・コードnをセット（n：マクロコード）
         TRAP    #1           マクロ命令の発行
         BNE     ERROR        マクロの処理結果を判定（Zビットがオンの
                              ときは正常終了）
           ⋮
ERROR    EQU     *            ┐
           ⋮                  ├エラー処理
                              ┘
PARAM    DS                   パラメータ格納エリア（mワード）（m：パラ
                              メータの格納に必要なワード数）
```

3.3.3 タ　ス　ク

（1）タスク番号

RMSは，RMSの下で実行するユーザ・プログラムを“タスク”という単位で管理する．このタスクの名前として番号を使用し，その番号を“タスク番号”と呼ぶ．タスク番号は，ユーザがシステム設計時に自由に決めることができ，プログラムをRMSに登録することにより番号付けされる．

タスク番号には次の規則がある．

- タスク番号は1～255で，1はRMSのシステム・タスクに割り付けられるので，ユーザは2～255のタスク番号を使用する．

- タスク番号は連続した番号である必要はなく，途中の番号が抜けていてもかまわない．
- タスク番号とそのプログラムの優先順位とは無関係である．

（2） タスクの優先度

タスクの処理の緊急度あるいは重要度に従ってタスクの優先度が決められる．いま，あるタスクの実行中にそれよりも優先度の高いタスクを起動する割込みがあった場合，RMS は実行中のタスクを一時中断して優先度の高いタスクの処理を実行し，その処理が終了した後で中断していたタスクの処理を再開する．

タスクの優先度は，1～255 の値で表わされ，これをタスクの“優先レベル (priority level)”と呼ぶ．優先レベルはその値が小さいほど優先度が高く，タスクを RMS に登録するときにユーザが自由に指定することができる．また，タスクを実行しているときでもマクロ命令 PLEVEL により変更できる．

図 3.7 に，タスクの優先レベルとその実行順序の例を示す．図 3.7 (a) は，実行可能な四つのタスク（タスク A, B, C, D）が実行を待っている状態を示しており，タスクの実行は，タスク A，B，C，D の順序で行なわれようとしている．図 3.7 (b) は，(a) の状態からタスク A の実行が終了し，新たにタスク E，F が実行可能となった状態を示す．(b) におけるタスクの実行順序は，タスク E，B，C，D，F の順序となる．†

（3） TIB (task information block) と TCB (task control block)

タスクを管理する情報は，TIB と TCB に格納される．TIB には，タスクのスタート番地，タスク番号など，タスク固有の情報が設定されており，ユーザがタスク登録時に設定しておかなければならない．TCB は，タスクの状態を管理するためのテーブルで，RMS は TIB をもとに TCB を RAM (random access memory) に作成し，この TCB を用いてタスクの実行制御を行なう．

（4） タスクの状態

i ） タスクの状態

タスクの状態には，次に示す六つの状態があり，各タスクは必ずどれか一つの状態に属すことになる．

† 優先レベル 1 は RMS のシステム・タスクで使用しており，ユーザ・タスクの優先レベルには 2～255 を割り付ける．また，複数のタスクに同じ優先レベルを割り付けてもかまわない．

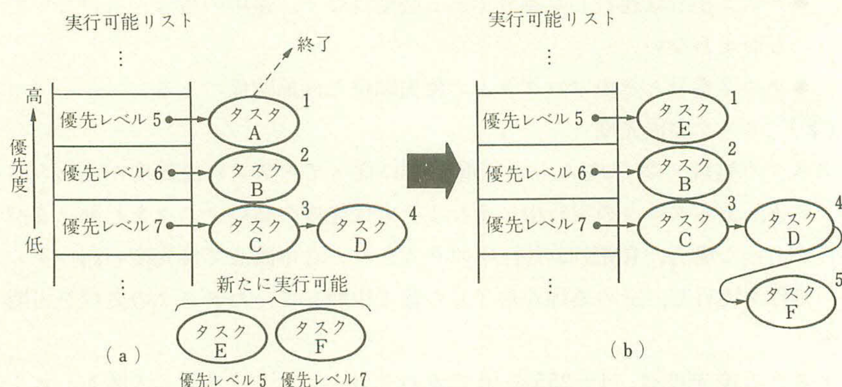


図 3.7　優先レベルと実行順序

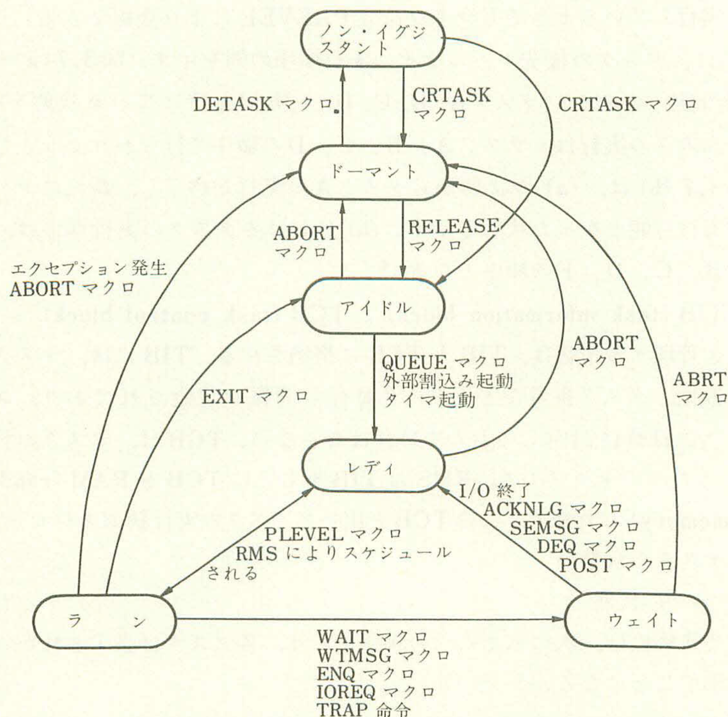


図 3.8　タスク状態の遷移

① アイドル (idle)：タスク起動待ち状態.

② レディ (ready)：タスク実行待ち状態. 起動がかかっていて, 実行待ちの状態をいう.

③ ラン (running)：タスク実行中. RMS は, レディ状態のタスクの中で最も優先レベルの高いタスクをラン状態とする. 最も優先レベルの高いタスクが複数個ある場合は, その中で最も早くレディ状態となったタスクをラン状態とする.

④ ウェイト (wait)：事象待ち状態. 事象の発生を待つため, タスクの実行が一時中断されている状態をいう.

⑤ ドーマント (dormant)：実行禁止状態.

⑥ ノン・イグジスタント (non-existent)：タスクが登録されていない状態.

ii) タスク状態の遷移

タスク状態の遷移を図 3.8 に示す. タスクは, 図に示すような状態をそれぞれのマクロ命令で遷移することにより, 動的に実行される.

3.3.4 タスク制御

(1) タスクの起動・実行・終了

i) タスクの起動

タスクの起動要因には, 次のものがある.

①外部事象によるタスクの起動

- プロセス割込みが発生した場合
- 入出力装置から指定したキーによるキー・イン割込みが発生した場合
- コンソールから QUEUE コマンドが入力された場合

②内部事象によるタスクの起動

- タスクからマクロ命令 QUEUE が発行された場合
- タイマで一定時間が経過した場合

起動されたタスクに制御を渡す前に, RMS は次の処理を行なう.

- スタック・ポインタ (A7) に, タスクが使用するスタック・エリアのアドレスをセットする.
- タスクの処理アドレスとして, TIB で指定されたタスクのスタート番地をセットする. 優先レベル, 割込みマスク・レベルは, TIB で指定され

た値をセットする．また，データ・レジスタD0には起動データをセットする．

- ステータス・レジスタのSビットを“0”にリセットする．すなわち，すべてのタスクはユーザ・モードで実行される．
- データ・レジスタD1にはタスク番号をセットする．

ii）タスクのスケジューリング

RMSは，レディ状態のタスクの中で最も優先レベルの高いタスクを実行させる．同一優先レベルの複数のタスクがレディ状態のときは，時間的に最も早くレディ状態となったタスクを最初に実行させる．

iii）タスクの中断・再開

タスク実行中に割込みが入ると，タスクの処理は中断され，割込みに関係する処理が終わって中断が解除されると，タスクは割込み前の状態を回復して処理を再開する．表3.11に，タスクの処理中断の要因とその期間を示す．

表 3.11　中断の要因と期間

項番	中断の要因		中断の期間
1	タスク自身から中断となる場合	IOREQマクロ	IOREQマクロ（入出力マクロ）命令を発行して入出力処理が終了するまで
		WAITマクロ	WAITマクロ命令を発行してから，POSTマクロまたは指定時間経過により中断解除されるまで
		ENQマクロ	要求した資源が割り付けられるまで
		WTMSGマクロ	メッセージが送られてくるまで
		TRAP 2～15	ACKNLGマクロ命令により中断解除されるまで
2	割込みにより中断させられる場合	プロセス割込み	プロセス割込みが発生して，プロセスの処理を終わってもとのタスクに戻ってくるまで
		入出力割込み	入出力装置からの終了割込みが発生し，入出力の処理を行ない，もとのタスクに戻ってくるまで
		タイマ割込み	タイマからの割込みが発生し，タイマ処理を行ない，もとのタスクに戻ってくるまで
		ブレーク条件	GOコマンドにより再開されるまで

iv）タスクの終了

タスクの処理が終了する要因としては，次のものがある．

①EXITマクロ命令

EXITマクロ命令を発行したタスクの処理は終了し，アイドル状態となる．

②ABORTマクロ命令

ABORTマクロ命令で指定されたタスクの処理は終了させられ，ドーマント

状態となる．

③エクセプション割込み

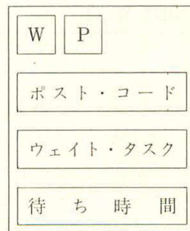
タスク実行中に，バス・エラー，アドレス・エラー，不当命令，ゼロによる割り算，CHK 命令，TRAPV 命令，特権違反，ライン 1010 および 1111 の割込みのいずれかが発生した場合は，それらの割込みに対する処理が指定されていなければ，そのタスクの実行を打ち切ってドーマント状態とする．

（2） タスク間の同期処理

タスク間の処理の同期をとるために"POST/WAIT"が使われる．この POST/WAIT 方式の中心となるのがイベント（event）である．イベントとはタスクにとって意味のある事象のことで，たとえば，一定時間が経過したとか，入出力処理が終了したとかがイベントとなる．イベントによるタスク間の同期を制御するために ECB (event control block) が設けられている．

i） ECB の構成

ECB は，各イベントごとにそのエリアをメモリ (RAM) に確保しなければならない．



ii） イベント発生の待ちと通知

イベント発生を連絡する側では，設定した ECB に対して POST マクロ命令を発行する．イベント発生の連絡を受ける側では，ECB に対して WAIT マクロ命令を発行する．

W ビット＝1 は，この ECB に対して WAIT マクロが発行されたことを示しており，P ビット＝1 は，この ECB に対して POST マクロが発行されたことを示している．ECB の初期状態では，W ビットおよび P ビットはともに 0 となる．

WAIT マクロを発行すると，P ビット＝0 ならば W ビット＝1 にし，WAIT マクロを発行したタスクはウェイト状態となる．P＝1 ならばすでに POST マ

クロが発行されているので，WAITマクロを発行したタスクはウェイト状態にはならず，タスクの処理は続行される．POSTマクロを発行すると，Wビット＝0ならばPビット＝1となる．Wビット＝1ならばWビット＝0にし，WAITマクロを発行したタスクをウェイト状態から解除する．また，WAITマクロで指定した待ち時間が経過してもPOSTマクロが発行されないときは，Wビット＝0にし，WAITマクロを発行したタスクをウェイト状態から解除する．

ECBのWビットおよびPビットのセット/リセットの関係を表3.12に示す．また，WビットあるいはPビット＝1のECBに対して，POSTあるいはWAITマクロ命令を発行したときの処理を表3.13に示す．

表 3.12　WビットおよびPビットのセット/リセット

項番	フラグ名	セット	リセット
1	Wビット	WAITマクロで行なう	POSTマクロまたは指定待ち時間経過により行なわれる
2	Pビット	POSTマクロで行なう	WAITマクロで行なう

表 3.13　POST/WAITマクロの処理内容

項番	ECBの状態	発行マクロ	処理内容
1	Wビット＝1	POSTマクロ	すでにWAITマクロを発行しているタスクのウェイト状態を解除し，タスクの優先順位に従って処理を再開する
		WAITマクロ	すでにWAITマクロが発行されているので，本マクロはエラーとなる
2	Pビット＝1	POSTマクロ	新しいポスト・コードをECBにセットする
		WAITマクロ	すでにPOSTマクロが発行されているので，ウェイト状態にならず処理は継続される

（3）　資源の共用管理

複数のタスクの非同期処理で必要となる機能に，前記の「タスク間の同期処理」のほかに，資源（リソース；resource）の共用管理がある．資源としては，メモリ，入出力機器，データ・セットおよびプログラムなどが考えられる．

不特定多数のタスク間で共通に参照あるいは更新するデータがある場合，あるタスクでそのデータを更新している間は，他のタスクからの参照および更新は禁止しなければならない．そうしなければ一方のタスクで更新中のデータを他のタスクが参照してしまうことになり，データの参照/更新が正しく行なわれないことになる．この参照/更新の制御を正しく行なうために“立て札”が

用いられている．共通のデータに対してそれぞれ立て札を用意し，データを更新しようとする場合にはこの立て札を見に行き，立て札が“空き”であれば立て札を“使用中”にして，データの更新処理を行なう．そして，データの更新が終わったら立て札をもとの“空き”に戻す．立て札が“使用中”のときは，別のタスクがデータを参照あるいは更新中であり，立て札が“空き”になるまで待たなければならない．

このように，タスク間で共通かつ逐次に使用する（あるタスクでの使用が終了してからでないと別のタスクが使用できない）資源に対する制御は，立て札を使って行なうことができる．立て札により制御を行なう資源としては，データ，プログラムおよび入出力装置などがあり，各立て札がどのデータあるいは入出力装置に対応するかは，使用するタスク間であらかじめ約束しておく必要がある．

RMSでは，以上の立て札に当たるものを資源管理テーブルRCB (resource control table) と呼ぶ．また，立て札を見に行き，“空き”を“使用中”にするのがENQマクロ命令であり，“使用中”であった立て札を“空き”に戻すのがDEQマクロ命令である．図3.9に，タスク間の共有データの制御の概念を示す．

i). 逐次再使用可能な資源とその利用

複数のタスクで参照するデータをあるタスクが更新する場合，そのデータの参照はデータを更新中のタスクのみに許し，他のタスクの参照は禁止しなければならない．このように，同時には1個のタスクしか使用できないプログラム（たとえば，実行中に自プログラムを書き換えるプログラム）などを逐次再使用可能な資源 (serially reusable resource) という．

複数のタスクが，ある資源を共用してその資源を逐次再使用した場合，その資源の使用を要求しているタスクの待ち行列を作り，待ち行列の先頭のタスクに資源を使用させる制御は，ENQマクロとDEQマクロの二つのマクロ命令で可能になる．

ii) 資源の番号付け

ユーザは，ENQマクロで指定する資源に対して1〜255の番号を付け，各資源の区別をしておく必要がある．各資源の区別は，すべて資源に付けられた番号によって行なわれるため，同一の資源をタスク間で共用する場合は番号を統

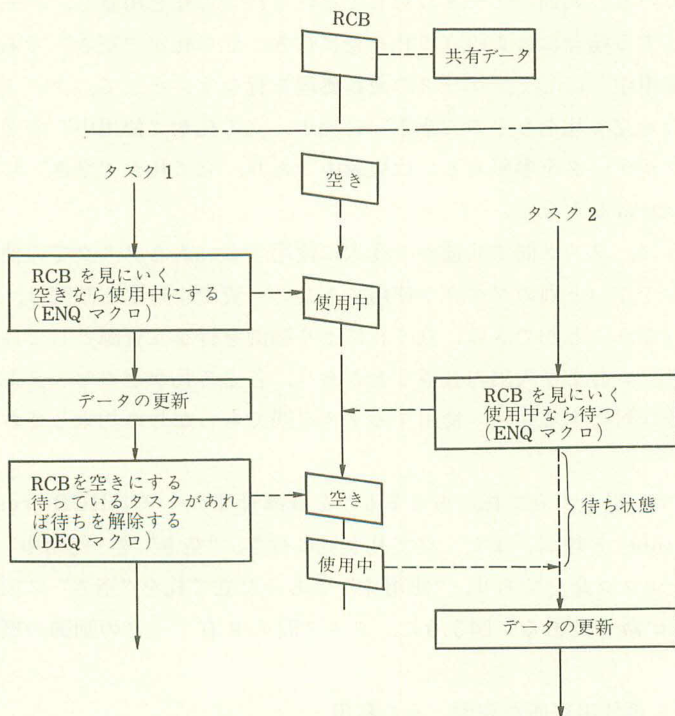


図 3.9　タスク間の共有データの制御

一して使用しなければならない．また，ENQおよびDEQマクロを使わずに直接資源を使用することも可能であるが，この場合，その資源に対する逐次使用の制御は保障されない．

ⅲ）資源の使用要求

資源の使用要求はENQマクロ命令で行なう．ENQマクロ発行時，タスクは次の三つの機能を選択することができる．

① 指定した資源がただちに使用可能かどうかのテストのみを行ない，資源の使用要求は行なわない．

② 指定した資源がただちに使用可能である場合にのみ資源の使用要求を行なう．ただちに使用可能でなければ資源の使用要求を取り下げ，発行元にリターンする．

③　指定した資源に対して無条件使用を要求し，ただちに使用できなければ，使用可能になるまで待つ．できるときはすぐ使う．

iv）　資源の解放

使用していた資源を解放するには，DEQ マクロを発行しなければならない．ENQ マクロで資源の使用要求を行なったタスクは，資源の使用が終わったら必ず資源の解放を行なわなければならない．

v）　使用要求待ち行列の制御

1個の逐次再使用可能な資源に対して複数のタスクから使用要求が発生した場合，使用要求の待ち行列が作られる．RMS が資源を割り当てていく順序は，ENQ マクロを発行した順序には関係なく，資源を要求したタスクの優先度に従って高い方から順番に資源が割り当てられていく．

（4）　タスク間のメッセージ通信

RMS では，可変長のメッセージをタスク間で送受信可能である．タスクは，SEMSG マクロ命令で必要なメッセージを他のタスクに送信し，REMSG マクロ命令で送信されてきたメッセージを取り込むことができる．

i）　メッセージ

メッセージのフォーマットを図3.10に示す．

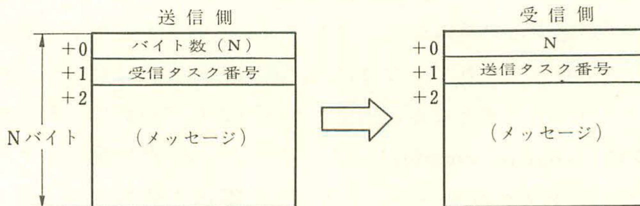


図 3.10　メッセージのフォーマット

ii）　メッセージ行列

タスクに送られてきたメッセージは，タスクに対応して作られたメッセージ行列につながれる．メッセージを受信するタスクは，あらかじめメッセージ行列に必要なエリアを確保し，TIB (task information block) で指定しておかなければならない．タスクから REMSG マクロ命令が発行されると，そのタスクのメッセージ行列の中から一番先に送られてきたメッセージが行列からはず

され，タスクに渡される．

ⅲ）　メッセージによる同期処理

タスクは，WTMSGマクロ命令を発行し，メッセージが送られてくるまで処理を一時中断することができる．WTMSGマクロ命令が発行されると，そのタスクのメッセージ行列が調べられ，メッセージが存在しない場合は，メッセージが送信されてくるまでタスクの処理が中断されることになる．メッセージが存在すれば，タスクの処理はそのまま継続される．

（5）　タスクの割込み処理

RMSには，タスク自身が発生させたエクセプション割込み（バス・エラー，アドレス・エラー，不当命令，ゼロによる割り算，CHK命令，TRAPV命令，特権違反，ライン1010および1111）やTRAP（TRAP 2～15）を処理できる機能がある．

ｉ）　エクセプション割込み処理

①ベクタの設定

タスクは，自分自身のエクセプション割込みに対するベクタをRMSに宣言することができる（EXVCTマクロ命令）．

②RMSの処理

タスク実行中にエクセプション割込みが発生すると，そのタスクがEXPVCTマクロ命令で対応する割込みのベクタを宣言しているかどうかが調べられる．対応するベクタが宣言されていなければ，タスクの実行は打ち切られ，タスクの状態はドーマント状態となる．ベクタが宣言されていれば，そのベクタ先をタスクのPC（program counter）としてタスクに制御を戻す．タスクに制御が戻ったとき，タスクのユーザ・スタックに割込み情報がセットされている（図3.11）．

ⅱ）　TRAP割込み処理

①ベクタの設定

タスクは，自分自身で発行したTRAP命令に対するベクタをTRPVCTマクロ命令で宣言することができる（ただし，TRAP 0，TRAP 1を除く）．

②RMSの処理

タスクがTRAP命令（TRAP 2～15）を実行すると，そのタスクがTRPVCTマクロ命令で対応するTRAPのベクタを宣言しているかどうかが調べられ

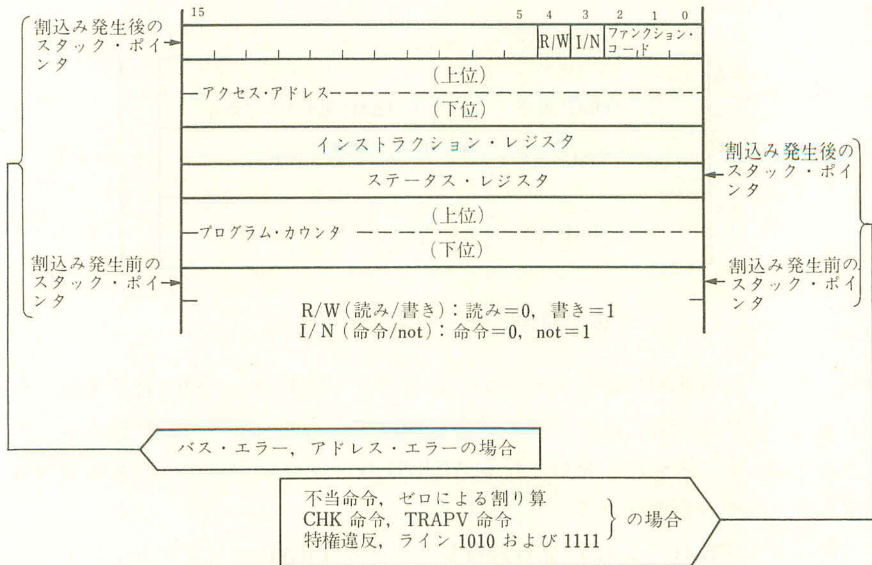


図 3.11 タスクのエクセプション割込み情報

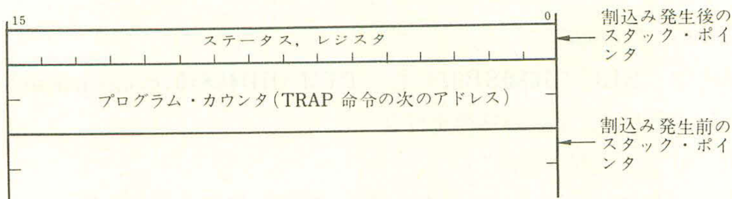


図 3.12 タスクのTRAP割込み情報

る.対応するベクタが宣言されていれば,そのベクタ先をタスクのPCとしてタスクに制御を戻す.そのとき,ユーザ・スタックに割込み情報がセットされている(図3.12).

(6) TRAP サービス

他タスクの発行したTRAP (TRAP 0, 1を除く)の処理をするタスクをサーバ・タスク (server task) という.タスクは,SERVE マクロ命令により特定のTRAPに対するサーバ・タスクとなることができる.

タスクからTRAP命令が発行され,かつそのTRAPに対するベクタがTRPVCTマクロ命令で宣言されていなければ,RMS は,そのTRAPのサー



図 3.13 サーバ・タスクへのメッセージ

バ・タスクにTRAP発生のメッセージを送り（図3.13），TRAPを発行したタスクの処理を中断し，サービス待ち状態にする．サービス待ち状態となったタスクは，他タスクから発行されるACKNLGマクロ命令により待ち状態が解除され，処理が再開される．

タスクのTRAP（2～15）発行に対して，そのTRAPのベクタおよびサーバ・タスクがともに存在しない場合は，RMSはデータ・レジスタD0にリターン・コード1をセットして，タスクに制御を戻す．

（7）　タイマ制御†

RMSは，SBC（H680SB01）上のPTM（HD46840 programmable timer module）を用いて下記の処理を行なう．

i）　日付，時刻の制御

50msの周期的なタイマ割込みを用い，時刻，日付を更新する．

ii）　ソフトウェア・タイマの制御

最大255個のソフトウェア・タイマを制御する．ソフトウェア・タイマは時間経過によるタスク起動に用いられ，1ms単位で時間を設定することができる．

iii）　入出力転送の処理時間を監視する．

iv）　WAITマクロ命令の待ち時間を監視する．

（8）　タスク制御マクロ命令

タスク制御マクロ命令には，次のようなものがある．

† PTMのタイマ1およびタイマ3はRMSで使用しているので，タイマ1，タイマ3に関するレジスタをアクセスしてはいけない．PTMは28ピンのMOS LSIである．

i) タスクの実行に関するマクロ

① QUEUE マクロ　　タスクの起動要求

② EXIT マクロ　　タスクの実行終了

③ ABORT マクロ　　タスクの実行禁止

④ RELEASE マクロ　　タスクの実行禁止解除

⑤ CRTASK マクロ　　タスクの登録

⑥ DETASK マクロ　　タスクの削除

ii) タスク間の同期に関するマクロ

① POST マクロ　　イベントの発生を連絡する

② WAIT マクロ　　イベントの発生を待つ

iii)逐次再使用可能資源の管理に関するマクロ

① END マクロ　　資源の占有要求

② DEQ マクロ　　資源の占有の解除

iv)メッセージ通信に関するマクロ

① SEMSG マクロ　　メッセージの送信

② WTMSG マクロ　　メッセージが送られてくるまで待つ

③ REMSG マクロ　　メッセージの読取り

④ CHMSGQ マクロ　　メッセージ行列の状態変更

v) 割込みに関するマクロ

① EXPVCT マクロ　　エクセプション割込みベクタの宣言

② TRPVCT マクロ　　TRAP 割込みベクタの宣言

vi) サーバ・タスクに関するマクロ

① SERVE マクロ　　サーバ・タスクの宣言

② ACKNLG マクロ　　サービス待ち状態の解除

vii) タイマに関するマクロ

① DTIME マクロ　　遅延タイマによるタスク起動

② CTIME マクロ　　遅延タイマのタスク起動の取消し

③ GTIME マクロ　　時刻の読出し

④ STIME マクロ　　時刻の設定

viii) その他のマクロ

① PLEVEL マクロ　　タスクの優先レベルの変更

② MLEVELマクロ　　割込みマスクのレベル変更

3.3.5 入出力制御

入出力制御は，ユーザが簡単に入出力処理を行なえるように，入出力装置の管理と，MPUと入出力装置間のデータの転送を行なう．

（1） 入出力制御の特徴

- RMSには，ACIA (HD46850 asynchronous communication interface adapter) およびプリンタ用PIA (HD46821 peripheral interface adapter) の入出力制御プログラムが標準プログラムとして組み込まれている．
- 上記以外の入出力制御プログラムは，EPROMに格納してRMSに組み込む必要がある．RMSに組み込んだ入出力制御プログラムは，入出力マクロ命令 (IOREQマクロ命令) からコールされる．入出力中に異常が発生した場合は，タスクやユーザのエラー処理ルーチンに異常が連絡される．

（2） 入出力制御の方法

i） I/O装置番号

各入出力装置を識別するために，接続される入出力装置のインタフェースごとにI/O装置番号が付けられる．I/O装置番号には，次の規則がある．

① I/O装置番号の範囲は1～255である．

② システム・タスク用のコンソール・タイプライタやプリンタの装置番号は，システム・テーブルで設定しなければならない．

③ 入出力装置とI/O装置番号との関係は，システム・テーブルで定義される．

ii） 入出力制御方式とRMSの動作

入出力制御時のMPUおよび入出力装置の動きを図3.14に示す．タスクmが入出力マクロ命令を発行すると，RMSは入出力制御を開始する．入出力制御の開始と同時に，タスクmは入出力データの転送が終了するまで入出力終了待ち (I/Oウェイトと呼ぶ) となる．タスクmがI/Oウェイト状態の間，RMSはほかに実行可能なタスクnにMPUを割り当て，処理を実行させる．

iii） 入出力装置使用中の制御

入出力装置使用中 (I/O BUSYという) とは，あるタスクが入出力要求中

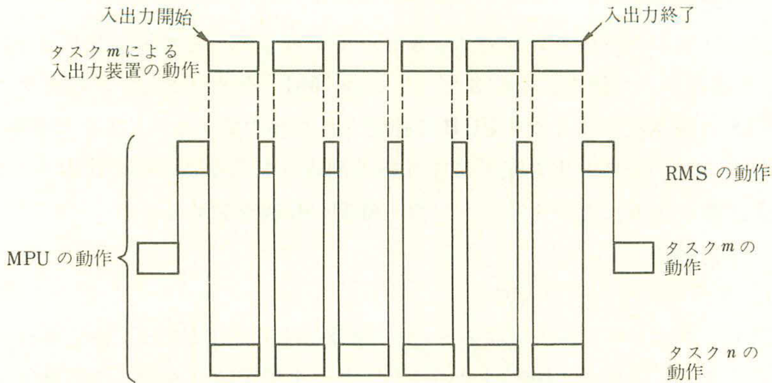


図 3.14　入出力装置とRMSの動作

に，他のタスクから同一入出力装置に対してさらに入出力要求があることをいう．RMS は，I/O BUSY ならばリターン・コードを9にしてユーザ・タスクに制御を戻す．I/O BUSY の場合，ユーザ・タスクは入出力要求の再試行が必要である．

iv）入出力のタイム・アウト

入出力を要求してから，それが終了したことを示す入出力装置からの応答があるまでの時間を監視し，ある一定時間以内に応答しない場合は，タイム・アウト (time out) のエラーを発生させる．

RMS では，各入出力装置のタイム・アウト時間を入出力別に秒単位で設定することができる．タイム・アウトが発生すると RMS は入出力転送を終了し，リターン・コードを11にしてユーザ・タスクに制御を戻す．

（3）標準 ACIA

RMS には ACIA の制御ルーチンがあり，ユーザはマクロ命令を用いてデータの入出力を行なうことができる．

i）ACIA の制御方法

ACIA を使用するためには，ACIA を定義するテーブル UCB (unit control block) を作成しておく必要がある．RMS は，ACIA に対する IOREQ マクロ命令を受け取ると，その ACIA に対応するテーブル UCB を参照し，ACIA の制御を行なう．

① 出力処理

ACIAコントロール・レジスタは8ビットであるが，その下位5ビットにはUCBで指定された値を，上位3ビットには“001”をセットして，出力データをACIAに書き込む．また，UCBで指定された出力時タイム・アウト時間の監視を始める．データの出力完了により終了割込みが入ると，次の出力データをACIAに書き込み，再びタイム・アウト時間の監視を始める．

② 入力処理

a） キーボードからの入力

ACIAのコントロール・レジスタの下位5ビットにはUCBで指定された値を，上位3ビットには“100”をセットし，データ受信による割込みの発生を待つ．この間，UCBで指定された入力時タイム・アウト時間の監視が行なわれる．データ受信の割込みが入ると，受信データの読込みが行なわれる．

b） 紙テープからの入力

ACIAのコントロール・レジスタの下位5ビットにはUCBで指定された値を，上位3ビットには“110”をセット（RTS＝high）し，紙テープ・リーダを駆動して，データ受信による割込み発生を待つ．また，UCBで指定された入力時タイム・アウト時間の監視を行なう．データ受信による割込みが発生すると，RTS＝lowにして受信データを読み込む．

③ 初期設定

RMSは，システム・イニシャライズ時ACIAに対して以下の処理を行なう．

a） コントロール・レジスタに＄03を書き込む（マスタ・リセット）．

b） コントロール・レジスタの下位5ビットにはUCBで指定された値を，上位3ビットには“100”をセットする（受信割込み待ち状態）．

④ アテンション割込み処理

入出力実行時以外に受信割込みが発生した場合は，受信データとUCBで指定されたアテンション・コードとの比較が行なわれる．受信データがアテンション・コードに等しければ，UCBで指定されたタスクを起動する．アテンション・コード以外のときは，割込みはクリアされ無視される．

ii） ACIAのエラー処理

ACIAの入出力時に発生するエラーには，フレーミング・エラー，オーバラン・エラー，パリティ・エラー，タイム・アウト・エラーがある．

RMSは，ACIAのエラーが発生すると次の処理を行なう．

① ACIAに$03をセットする（マスタ・リセット）．

② ACIAのコントロール・レジスタの下位5ビットにはUCBで指定された値を，上位3ビットには“100”をセットする．

③ ユーザのエラー処理ルーチンがUCBで指定されていれば，エラー処理ルーチンをコールする．

④ I/Oウェイト状態のタスクにエラー・コードを渡し，タスクの状態をレディ状態とする（ユーザのエラー処理ルーチンのリターン・パラメータでリトライの指定があれば，再度入出力転送を行なう）．

（4） 標準プリンタ

RMSにはPIAを介して制御するプリンタ（セントロニクス・インタフェース仕様）の制御ルーチンがあり，ユーザはマクロ命令を用いてプリンタへの出力を行なうことができる．

i） プリンタの制御方法

① ハードウェア・インタフェース

MPUから見たPIAのプリンタ・インタフェース仕様を表3.14に示す．

② 出力処理

プリンタに対する出力は下記手順で行なう．

a） プリンタの状態（エラー）をチェックする．

b） プリンタの状態がBUSYならば，PIAのAポート・コントロール・レ

表 3.14 PIAプリンタ・インタフェース

信号名	PIA端子	IN/OUT	信号名	PIA端子	IN/OUT
DATA 1	PA 0	OUT	SEL	PB 0	IN
DATA 2	PA 1	OUT	PE	PB 1	IN
DATA 3	PA 2	OUT	BUSY	PB 2	IN
DATA 4	PA 3	OUT	(not use)	PB 3	−
DATA 5	PA 4	OUT	(not use)	PB 4	−
DATA 6	PA 5	OUT	(not use)	PB 5	−
DATA 7	PA 6	OUT	(not use)	PB 6	−
DATA 8	PA 7	OUT	(not use)	PB 7	−
$\overline{\text{ACKNLG}}$	CA 1	IN	FAULT	CB 1	IN
DATA STB	CA 2	OUT	INPUT PRIME	CB 2	OUT

PB3～PB7は未使用であるが，モニタ・ボードでは他の目的で使用しているため，ユーザは使用できない

ジスタに$3Dをセットし，プリンタからのACKNLG信号による割込みを待つ．また，UCBで指定されたタイム・アウト時間の監視を始める．

c）　プリンタがBUSYでなければ，出力データを書き込み，DATA STB信号を送る．DATA STB信号は，データを書き込み2マイクロ秒経過後，2マイクロ秒間出力する．

③　リセット処理

システム・イニシャライズ時，RMSはプリンタ用PIAに対して次の処理を行なう．

a）　PIAのAポートおよびBポートのコントロール・レジスタに$00をセットする．

b）　PIAのAポート・データ・ディレクション・レジスタに$FFをセットし，Aポートを出力ポートとする．

c）　PIAのBポート・データ・ディレクション・レジスタに$00をセットし，Bポートを入力ポートとする．

d）　PIAのAポートおよびBポートのコントロール・レジスタに$3Cをセットする．

ii）　プリンタのエラー処理

プリンタのエラーには，ノット・セレクト，ペーパ・エンプティ，タイム・アウト・エラーがある．

エラーを検出すると，RMSは次の処理を行なう．

①　ユーザのエラー処理ルーチンがUCBで指定されているかどうかを調べ，指定されていればエラー処理ルーチンをコールする．

②　I/Oウェイト状態のタスクにエラー・コードを渡し，タスクの状態をレディ状態とする（ユーザのエラー処理ルーチンのリターン・パラメータでリトライの指定があれば，再度入出力転送を行なう）．

（5）　拡張入出力装置

ACIA，プリンタ以外の入出力装置に対する制御プログラムは，ユーザが作成して，RMSに組み込まなければならない．

i）　処理の流れ

①　ユーザ・ルーチンのコール

RMSは，次の三つの要因によりユーザの入出力制御ルーチンに制御を渡す

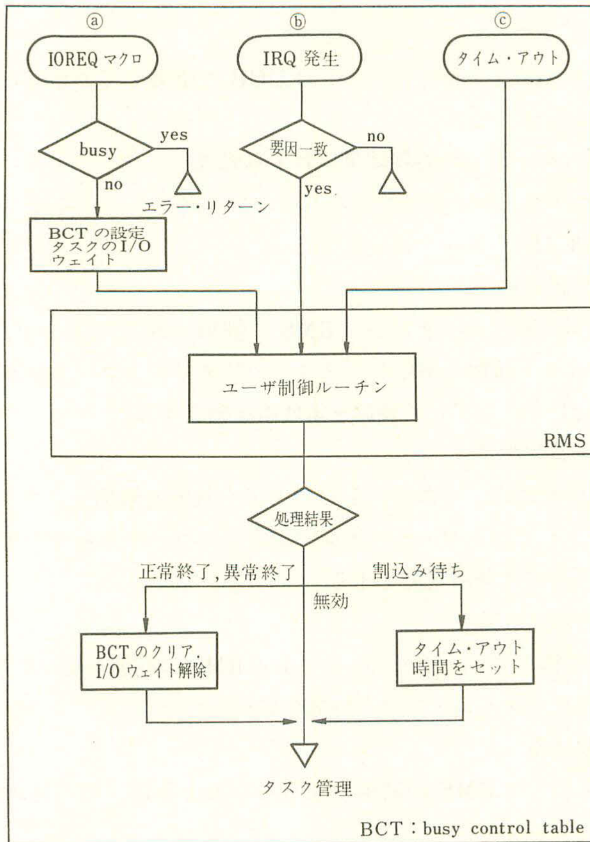


図 3.15 拡張入出力制御の処理フロー

(図3.15).

a) タスクから IOREQ マクロ命令が発行されたとき

- その装置が使用中かどうかを BCT (busy control table) で調べ, 使用中であればリターン・コード 9 をセットしてタスクにリターンする.
- 使用中でなければ, タスクを I/O ウェイト状態にして, ユーザの入出力制御ルーチンに制御を渡す.

b) IRQ 割込み分析により要因が一致したとき, ユーザ・ルーチンに制御を渡す.

c）　タイム・アウトのエラーが発生すると，ユーザ・ルーチンに制御を渡す．

なお，以上三つのユーザ・ルーチンはUCBで定義しておかなければならない．

②　ユーザ・ルーチンの処理結果に対する処理

RMSは，ユーザ・ルーチンからのリターン・パラメータに従って次の処理を行なう（図3.15）．

a）　入出力正常終了

正常終了でユーザ・ルーチンからRMSに制御が戻ってくると，RMSはタスクのI/Oウェイト状態を解除し，タスクのリターン・コードに0をセットする．また，BCTをクリアして装置を未使用状態とする．

b）　入出力異常終了

異常終了の場合には，タスクのI/Oウェイト状態を解除し，タスクへのリターン・コードとしてユーザ・ルーチンで指定したコードをセットする．また，BCTをクリアして未使用状態とする．

c）　入出力割込み待ち

入出力割込み待ちでユーザ・ルーチンからRMSに制御が戻ってきた場合は，UCBで指定されたタイム・アウト時間の監視を行なう．

d）　無効割込み

無効割込みとしてRMSに制御が戻ってきたときは，何の処理も行なわない．

3.4　アセンブリ言語

68000のアセンブラとしては2種類が（株）日立製作所から提供されている．汎用計算機（HITAC-MシリーズやIBM S/370，303Xなど）で動作するクロス・マクロ・アセンブラと，68000用システム開発装置H680 SD300で動作するレジデント・マクロ・アセンブラの2種類がそれである．

以下では，この2種類のアセンブラに共通する内容を中心に，アセンブリ言語について説明する．

3.4.1 アセンブリ言語の基本要素

(1) 文字セット

ソース・プログラムをコーディングするのに使える文字として，次のような文字がある．基本的には，ASCII (American standard code for information interchange) の $20〜$5F のすべての文字が使える．

i) 英　字　A, B, C, …, Z の26文字

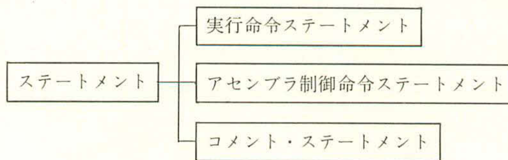
ii) 数　字　0, 1, 2, …, 9 の10文字

iii) 特殊文字　(空白), !, ”, #, $, %, &, ′, (,), *, +, , ,
−, . , /, :, ;, <, =, >, ?, @, \

(2) ステートメント

i) ステートメントの種類

コーディングされたソース・プログラムは，一連のソース・ステートメントから構成されており，個々のステートメントは次の3種類に分けられる．



① 実行命令ステートメント

実行命令ステートメントは，68000 の機械語命令と1対1に対応し，あらかじめ定義されたニモニック (mnemonic) とフォーマットに従ってコーディングしなければならない．実行命令ステートメントは，アセンブラにより 2〜10 バイトの機械語命令に変換される．

② アセンブラ制御命令ステートメント

アセンブラ制御命令ステートメントは，主にアセンブラの処理を制御するために使われる．このアセンブラ制御命令の中では，データ定義のための制御命令だけが機械語に変換される．アセンブラ制御命令には，データの定義および確保，メモリ領域の確保などを行なう命令がある．

③ コメント・ステートメント

コメント・ステートメントは，プログラムの理解を助け，プログラムのデバッグと保守を容易に行なえるようにする目的で使われる．コメント・ステートメントはアセンブラにより機械語には変換されず，アセンブル・リストに出力

されるだけである．

コメントの書き方には，次の二つの方法がある．

a）ステートメントの先頭（第1カラム）にアスタリスク（＊）を書くと，ステートメント全体がコメント（すなわちコメント・ステートメント）になる．

b）実行命令ステートメントおよびアセンラ制御命令ステートメントのオペランド・フィールド（または，オペレーション・フィールド）の後に，1文字以上のスペースを空けて書く．

【例1】

```
  ┌─第1カラム
* THIS ENTIRE LINE IS A COMMENT ←─コメント・ステートメント
  BRA LABEL    THIS COMMENT FOLLOWS AN INSTRUCTION.
  ─────────    ────────────────────────────────────
      ↑                        ↑
   実行命令                  コメント
```

ii）ステートメントのフォーマット

コメント・ステートメントを除く実行命令ステートメントおよび制御命令ステートメントは，ラベル・フィールド，オペレーション・フィールド，オペランド・フィールド，コメント・フィールドの四つのフィールドから構成されている．

ラベル・フィールド	オペレーション・フィールド	オペランド・フィールド	コメント・フィールド

ステートメントの記述はフリー・フォーマットであり，したがって各フィールド間は必ず1文字以上のスペースで区切る必要がある．

① ラベル・フィールド

ラベル・フィールドには，命令で参照するラベルを書く．ラベルは

- 第1カラムから始まり，スペースで終わるシンボル
- 任意のカラムから始まり，コロン（：）で終わるシンボル

のいずれかである．

また，アセンブラの予約語である次のシンボルは，ラベルとしては使えない．

- データ・レジスタ　　D0，D1，D2，…，D7
- アドレス・レジスタ　A0，A1，A2，…，A7
- その他のレジスタ　　CCR，SR，SP，USP

② オペレーション・フィールド

オペレーション・フィールドには，実行命令，アセンブラ制御命令，またはマクロ・コール命令 (macro call) のニモニックを書く．

命令によって処理されるデータのサイズは，オペレーションのニモニックの後に続けて，ピリオド（．）とともにデータ・サイズ・コードを併記することによって指定される．

.B	バイト・データ
.W	ワード・データ
.L	ロング・ワード・データ
.S	ショート（B_{CC}，BSR命令に用いる）

異なるサイズのデータの処理が可能な命令は，必ずデータのサイズをデータ・サイズ・コードで指定する必要がある．指定を行なわない場合は，デフォールト (default) のワードがデータ・サイズとして仮定される．処理を行なうデータのサイズが決まっている命令の場合には，データ・サイズの指定は必要ない（指定してもしなくてもどちらでもよいが，誤ったデータ・サイズを指定した場合はアセンブル・エラーとなる）．

【例2】

```
MOVE.B    D0, D1    バイト・データの転送
MOVE.W    D0, D1    ワード・データの転送  ┐同じ処理
MOVE      D0, D1    ワード・データの転送  ┘
    ↑
    └── デフォールト（ワード）
MOVE.L    D0, D1    ロング・ワード・データの転送
```

【例3】

```
LEA      ADDR, A0    アドレス（ロング・ワード）のロード ┐同じ処理
LEA.L    ADDR, A0    アドレス（ロング・ワード）のロード ┘
    ↑
    └── 指定してもしなくてもどちらでもよい
```

③ オペランド・フィールド

オペランド・フィールドには，オペレーション・フィールドの命令に対応するオペランドを書く．命令によっては，オペランドが必要でないものや，複数のオペランドを必要とするものがある．

④ コメント・フィールド

コメント・フィールドはアセンブル・リスト出力時のみ有効で，命令の処理の説明などのコメントを書くのに使う．

（3）　データ形式

アセンブリ言語で記述できるデータ形式には，数値定数，文字定数，シンボル，式の4種類がある．

i）　数値定数

数値定数としては，10進数のほか，16進数，8進数†，2進数がある．

①　10進数

10進数は0，1，2，…，9の数字で表わされ，特に指定のない数は10進数となる．

②　16進数

16進数は0，1，2，…，9，A，B，…，Fで表わされ，先頭にドル記号（$）を付けて16進数であることを示す．

③　8進数

8進数は0，1，2，…，7の数字で表わされ，先頭にアットマーク（@）を付けて8進数であることを示す．

④　2進数

2進数は0，1の数字で表わされ，先頭にパーセント記号（%）を付けて2進数であることを示す．

【例4】

```
10進数    4096
16進数    $1000（10進で4096）
 8進数    @10000（10進で4096）
 2進数    %1000000000000（10進で4096）
```

ii）　文字定数（リテラル）

文字定数はアポストロフィ（'）で文字列を囲んで表わす．文字列の各文字は7ビットのASCIIコードに変換される．文字定数の中にアポストロフィを含む場合は，アポストロフィを二つ続けて書かなければならない．

【例5】

```
DC.L    'ABCD'        ロング・ワード・データ$41424344をメモリに確保
DC.L    ''''''''''    ロング・ワード・データ$27272727をメモリに確保
DC      '*'           ワード・データ$2A00をメモリに確保（左づめにセット）
DC.W    '* '          ワード・データ$2A20をメモリに確保
```

---

†　8進数はレジデント・マクロ・アセンブラのみ使用可能．

iii) シンボル

① シンボルの規則

シンボルは，英字またはピリオド(.)で始まる30文字以内の英数字またはドル記号($)の文字列である(クロス・マクロ・アセンブラとレジデント・マクロ・アセンブラで若干仕様が異なるので注意が必要).

アスタリスク(*)をシンボルとして用いた場合には，ロケーション・カウンタ(location counter)を意味しており，*をオペランド・フィールドにもつ命令の先頭バイトのアドレスを示す.

【例6】 次の二つの命令は同じ意味になる.

```
        BRA   *   ←ロケーション・カウンタ
LOOP:   BRA   LOOP
```

② 絶対シンボルと相対シンボル

シンボルはそのシンボルのもつ値の属性により二つに分けられる．一つは絶対シンボルであり，他の一つは相対シンボルである．相対シンボルはリロケータブルなセクションで定義されたシンボルで，その値はセクションの先頭からの相対番地が割り当てられる．したがって，相対シンボルのもつ値の属性は相対値となる．これに対し，絶対シンボルはアブソリュートなセクションで定義されたシンボル，または，アセンブラ制御命令のEQU，SETで定数と結ばれたシンボルで，その値の属性は絶対値となる.

シンボルのもつ値の属性は，アドレス形式に対して重要な意味をもつばかりでなく，いくつかのアセンブラ制御命令や式の評価においても重要な意味をもっている.

【例7】

```
        ORG    $100            アブソリュートなセクションの始まり
ABS1    EQU    *               絶対シンボル
ABS2    EQU    $20             絶対シンボル
        NOP
        SCT    1               リロケータブルなセクションの始まり
        DS     5
REL1    EQU    *               相対シンボル
        SCT    2               リロケータブルなセクションの始まり
        MOVE   ABS 1(A0), D0
        MOVE   ABS 2(A1), D1
        MOVE   REL 1(A2), D2
        END    ↑この指定はアセンブル・エラーとなる
```

iv）式

式はシンボル，数値定数，文字定数および演算子から構成される．

① 演算子

式の中で用いられる演算子には，算術演算子，シフト演算子および論理演算子がある．

a）算術演算子

加算 ＋，減算 －，乗算 ＊，除算[†]／，符号のマイナス －

b）シフト演算子

右シフト　≫　指定されたビット数だけ右にシフト

左シフト　≪　指定されたビット数だけ左にシフト

【例8】

```
HH   EQU   %0101≫1    HH EQU %0010に同じ
II   EQU   %1111≪2    II UQU %1100に同じ
```

c）論理演算子

論理積　&　，論理和　！

【例9】

```
HH   EQU   %0101&%1111    HH EQU %0101に同じ
II   EQU   %0101!%1111    II EQU %1111に同じ
```

式は，次に示す優先順位に従って計算が行なわれる．

1）（　）で囲まれた式（内側をより優先）

2）符号のマイナス

3）シフト

4）論理積，論理和

5）乗除算

6）加減算

優先順位の等しい演算子は左から右の順に計算が行なわれる．式の計算の結果（途中結果も含めて）は，すべて32ビットで表わされる整数である．

② 絶対式と相対式

シンボル同様，式にもその式の値の属性から絶対式（絶対シンボル，定数を含む）と相対式（相対シンボルを含む）がある．相対式に使える演算子は，＋

† 除算の場合，小数点以下を切り捨てた整数の値が結果となる．

と一に限られる．

【例10】

```
        ORG   $100         アブソリュートなセクションの始まり
ABS1    EQU   *
ABS2    EQU   $20
        NOP
        SCT   1            リロケータブルなセクションの始まり
        DS    5
REL1    EQU   *
REL2    EQU   *+$30
        SCT   2            リロケータブルなセクションの始まり
SYM1    EQU   REL1+ABS1    相対式
SYM2    EQU   REL2-ABS2    相対式
SYM3    EQU   REL2-REL1    絶対式
SYM4    EQU   `ABC'+2      絶対式
SYM5    EQU   REL1*2
        END      \________アセンブル・エラーとなる
```

3.4.2 アドレシング・モードの表記法

各種アドレシング・モードについてのアセンブラの表記法を示す．

（1） レジスタ直接アドレシング

i） データ・レジスタ直接

表記法：Dn （nは0～7）

【例1】

```
CLR.L   D1   D1全体（32ビット）をクリアする
```

ii） アドレス・レジスタ直接

表記法：An （nは0～7）

【例2】

```
ADD   A1,   A2   A1の下位ワードをA2の下位ワードに加え，結果をA2へ格納する
```

（2） アドレス・レジスタ間接アドレシング

i） アドレス・レジスタ間接

表記法：(An) （nは0～7）

【例3】

```
MOVE   #5, (A5)   A5の内容が示す番地のメモリ（ワード）へ5をセットする
```

SUB.L	(A1), D0	D0 から A1 の内容が示す番地のメモリ（ロング・ワード）の値を減算し，結果を D0 に格納する

ii）ポストインクリメント・アドレス・レジスタ間接

表記法：(An)＋　（n は 0〜7）

【例 4】

MOVE.B	(A2)＋, D2	A2 の内容が示す番地のメモリ（バイト）の内容を D2 へロード，さらに A2 の内容に 1 が加算
MOVE.L	(A4)＋, D3	A4 の内容が示す番地のメモリ（ロング・ワード）の内容を D3 へロード，さらに A4 の内容に 4 が加算

iii）プリデクリメント・アドレス・レジスタ間接

表記法：−(An)　（n は 0〜7）

【例 5】

CLR	−(A2)	A2 から 2 を減算し，その結果が示す番地のメモリ（ワード）の内容をクリア
CMP.L	−(A0), D0	A0 から 4 を減算し，その結果が示す番地のメモリ（ロング・ワード）の内容と D0 の内容を比較

iv）ディスプレースメント付きアドレス・レジスタ間接

表記法：＜式＞(An)（式は絶対式，n は 0〜7）

【例 6】

AVAL	EQU	5	
	CLR.B	AVAL(A0)	A0 の内容プラス 5 番地のメモリ（バイト）の内容をクリア
	MOVE.W	#2, 10(A2)	A2 の内容プラス 10 番地のメモリ（ワード）へ 2 をセット

v）インデックス付きアドレス・レジスタ間接

表記法：＜式＞（An, Ri.W）
　　　　＜式＞（An, Ri.L）
（式は絶対式，n は 0〜7，Ri は D0〜D7，A0〜A7，.W は省略可能．）

【例 7】

ADD	10(A1, D2), D5	A1 の内容とインデックス・レジスタ D2 の下位ワードの内容の和プラス 10 番地のメモリ（ワード）の内容を D5 の下位ワードへ加え，結果を D5 に格納する
MOVE.L	D5, $20(A2, A3, L)	A2 の内容とインデックス・レジスタ A3 の内容の和プラス $20 番地のメモリ（ロング・ワード）へ D5 の内容をセットする

（3） アブソリュート・アドレシング

i） アブソリュート・ショート

表記法：×××× （××××は＄0000～＄7FFF，＄FFFF8000～＄FFFFFFFF）

【例8】

```
JMP    $4000    $4000番地へジャンプ
```

ii） アブソリュート・ロング

表記法：××××××××

【例9】

```
JMP    $12000    $12000番地へジャンプ
```

（4） プログラム・カウンタ相対アドレシング

i） ディスプレースメント付きプログラム・カウンタ相対

表記法：＊±＜式＞ （式は絶対式）
＜ラベル＞

【例10】

```
BGT    *+$10    条件が成立すればPCプラス$10番地へブランチ
BEQ    RSYM     条件が成立すればRSYMへブランチ
```

ii） インデックス付きプログラム・カウンタ相対

表記法：＜ラベル＞（Ri.W） （RiはD0～D7，A0～A7
＜ラベル＞（Ri.L） .Wは省略可能）

【例11】

```
MOVE    T(D2), TABLE    TとD2の下位ワードの内容の和が示す番地のワー
                        ド・データをTABLEへセット
```

（5） イミディエート・データ・アドレシング

i） イミディエートおよびクイック・イミディエート

表記法：#＜データ＞

【例12】

```
MOVE       #1, D0      D0の下位ワードへ1をロードする
CMPI.B     #'?', D0    D0の最下位バイトの内容を比較する
ADDQ.B     #1, (A0)    A0の示す番地のメモリ（バイト）の内容に1を加算する
MOVEQ.L    #1, D0      D0に1をロードする
```

ii）コンディション・コードとステータス・レジスタ

表記法：CCR

　　　　SR

【例13】

```
ORI.B      #$1F,CCR      CCRの全コンディションをオンにする
ANDI.B     #$00, CCR     CCRの全コンディションをクリアする
MOVE.W     #$2700, SR    スーパバイザ状態，割込みマスクのセットする
```

以上のアドレス形式をまとめて表3.15に示す．

表 3.15　アドレス形式

アドレシング・モード	アドレス表記法	備　考
レジスタ直接アドレシング		nは0～7
データ・レジスタ直接	Dn	
アドレス・レジスタ直接	An	
アドレス・レジスタ間接アドレシング		nは0～7，iは0～7
アドレス・レジスタ間接	(An)	
ポストインクリメント・アドレス・レジスタ間接	(An)+	
プリデクリメント・アドレス・レジスタ間接	−(An)	
ディスプレースメント付きアドレス・レジスタ間接	＜式＞（An）	式の値は符号付き16ビット
インデックス付きアドレス・レジスタ間接	＜式＞（An,Ri,W） ＜式＞（An,Ri,L）	式の値は符号付き8ビット
アブソリュート・アドレシング		
アブソリュート・ショート	××××	
アブソリュート・ロング	××××××××	
プログラム・カウンタ相対アドレシング		iは0～7
ディスプレースメント付きプログラム・カウンタ相対	*±＜式＞ ＜ラベル＞	ディスプレースメントは符号付き16ビット
インデックス付きプログラム・カウンタ相対	＜ラベル＞（Ri,W） ＜ラベル＞（Ri,L）	ディスプレースメントは符号付き8ビット
イミディエート・データ・アドレシング		
イミディエートおよびクイック・イミディエート	#＜データ＞	
コンディション・コードとステータス・レジスタ	CCR SR	

3.4.3　実　行　命　令

（1）実行命令の変化形

実行命令の中には，“アドレス”あるいは“イミディエート”といった命令の

変化形をとるものがある（表2.4の命令を参照）.

この中で，オペランドのアドレシング・モードに従って，アセンブラが自動的に変化形を選択する命令がある．たとえば

```
                 ┌──── イミディエート・データ（－128～＋127）
                 ↓
MOVE.L        #100, D0
──────        ────  ──
  ↑                  ↑
  └── ロング・        └── データ・レジスタ直接
      ワード
```

という実行命令のコーディングは，アセンブラにより

MOVEQ　　#100，D0

の機械語に自動的にアセンブルされる．このように，アセンブラが自動的に変化形を選択する命令とそのオペランドのアドレス形式を，表3.16に示す．

表 3.16 実行命令の変化形

命令	変化形	オペランドのアドレス形式		備考
		ソース	デスティネーション	
MOVE	MOVEA	—	An	バイト・サイズは不可
	MOVEQ	#＜データ＞	Dn	データは－128～＋127
ADD	ADDA	クイック以外	An	バイト・サイズは不可
	ADDI	#＜データ＞	Dn，An以外	
	ADDQ	#＜データ＞	—	データは1～8
SUB	SUBA	クイック以外	An	バイト・サイズは不可
	SUBI	#＜データ＞	Dn，An以外	
	SUBQ	#＜データ＞	—	データは1～8
CMP	CMPA	—	An	バイト・サイズは不可
	CMPI	#＜データ＞	Dn，An以外	
	CMPM	(An)+	(An)+	
AND	ANDI	#＜データ＞	Dn以外	
OR	ORI	#＜データ＞	Dn以外	
EOR	EORI	#＜データ＞	—	

（2） オペランドのフォーマット

以下では，命令の種別ごとにオペランドのフォーマットを示す，表3.17～表3.24で使われている記号の意味は，次の通りである．

Dn：データ・レジスタ（D0～D7）

An：アドレス・レジスタ（A0～A7）

Rn：データ・レジスタまたはアドレス・レジスタ

＜ea＞：指定可能な実効アドレス

USP：ユーザ・スタック・ポインタ

SR：ステータス・レジスタ

CCR：コンディション・コード・レジスタ

d，＜displacement＞：ディスプレースメント
#＜data＞，#＜displacement＞，#＜vector＞：イミディエート・データ
＜label＞：ラベル

*：処理の結果に従ってセット
－：処理による影響なし
0：クリア
1：セット
U：未定数（不確定）

（以上 コンディション・コード）

i）データ転送命令

表3.17に，データ転送命令のオペランドのフォーマットを示す．

表 3.17　データ転送命令のオペランドのフォーマット

命　令	サイズ			オペランドのフォーマット	コンディション・コード					備　　考
	B	W	L		X	N	Z	V	C	
EXG			○	Rx, Ry	－	－	－	－	－	
LEA			○	＜ea＞, An	－	－	－	－	－	
LINK	－	－	－	An, #＜displacement＞	－	－	－	－	－	UNLKと対をなす
MOVE	○	○	○	＜ea＞, ＜ea＞	－	*	*	0	0	Anモードのときバイトは不可
MOVEA		○	○	＜ea＞, An	－	－	－	－	－	
MOVEM		○	○	＜register-list＞, ＜ea＞	－	－	－	－	－	－(An)モード可
		○	○	＜ea＞, ＜register-list＞	－	－	－	－	－	(An)＋モード可
MOVEP		○	○	Dx, d(Ay)	－	－	－	－	－	
		○	○	d(Ay), Dx	－	－	－	－	－	
MOVEQ			○	#＜data＞, Dn	－	*	*	0	0	－128≦＜data＞≦＋127
PEA			○	＜ea＞	－	－	－	－	－	
SWAP		○		Dn	－	*	*	0	0	
UNLK	－	－	－	An	－	－	－	－	－	LINKと対をなす

MOVEM命令の＜register-list＞の表記法は

D0／D1／D2／A0／A1／A3

（↑ データ・レジスタD0, D2, D3とアドレス・レジスタA0, A1, A2が処理の対象となることを意味する）

または

D0－D2／A0－A2

（↑ 意味は上の表記法と同じ）

となる．

ii) 整数算術演算命令

表3.18に，整数算術演算命令のオペランドのフォーマットを示す．

表 3.18 整数算術演算命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
ADD	○	○	○	<ea>, Dn	*	*	*	*	*	Anモードのときバイトは不可
	○	○	○	Dn, <ea>	*	*	*	*	*	AnモードのときはADDA
ADDA		○	○	<ea>, An	—	—	—	—	—	
ADDI	○	○	○	#<data>, <ea>	*	*	*	*	*	AnモードのときはADDA
ADDQ	○	○	○	#<data>, <ea>	*	*	*	*	*	1≦<data>≦8
ADDX	○	○	○	Dx, Dy	*	*	*	*	*	
	○	○	○	−(Ay), −(Ax)	*	*	*	*	*	
CLR	○	○	○	<ea>	—	0	1	0	0	
CMP	○	○	○	<ea>, Dn	—	*	*	*	*	Anモードのときバイトは不可
CMPA		○	○	<ea>, An	—	*	*	*	*	
CMPI	○	○	○	#<data>, <ea>	—	*	*	*	*	AnモードのときはCMPA
CMPM	○	○	○	(Ay)+, (Ax)+	—	*	*	*	*	
DIVS		○		<ea>, Dn	—	*	*	*	0	符号付き
DIVU		○		<ea>, Dn	—	*	*	*	0	符号なし
EXT		○	○	Dn	—	*	*	0	0	
MULS		○		<ea>, Dn	—	*	*	0	0	符号付き
MULU		○		<ea>, Dn	—	*	*	0	0	符号なし
NEG	○	○	○	<ea>	*	*	*	*	*	
NEGX	○	○	○	<ea>	*	*	*	*	*	
SUB	○	○	○	<ea>, Dn	*	*	*	*	*	Anモードのときバイトは不可
	○	○	○	Dn, <ea>	*	*	*	*	*	AnモードのときはSUBA
SUBA		○	○	<ea>, An	—	—	—	—	—	
SUBI	○	○	○	#<data>, <ea>	*	*	*	*	*	An モードのときはSUBA
SUBQ	○	○	○	#<data>, <ea>	*	*	*	*	*	1≦<data>≦8
SUBX	○	○	○	Dx, Dy	*	*	*	*	*	
	○	○	○	−(Ay), −(Ax)	*	*	*	*	*	
TAS	○			<ea>	—	*	*	0	0	
TST	○	○	○	<ea>	—	*	*	0	0	

iii) 論理演算命令

表3.19に，論理演算命令のオペランドのフォーマットを示す．論理演算命令では，アドレス・レジスタに対する論理演算ができないので注意が必要である．

表 3.19　論理演算命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
AND	○	○	○	<ea>, Dn	−	*	*	0	0	
	○	○	○	Dn, <ea>	−	*	*	0	0	
ANDI	○	○	○	#<data>, <ea>	−	*	*	0	0	SR モード可
EOR	○	○	○	Dn, <ea>	−	*	*	0	0	
EORI	○	○	○	#<data>, <ea>	−	*	*	0	0	SR モード可
NOT	○	○	○	<ea>	−	*	*	0	0	
OR	○	○	○	<ea>, Dn	−	*	*	0	0	
	○	○	○	Dn, <ea>	−	*	*	0	0	
ORI	○	○	○	#<data>, <ea>	−	*	*	0	0	SR モード可

表 3.20　シフトおよびローテート命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
ASL	○	○	○	Dx, Dy	*	*	*	*	*	
	○	○	○	#<data>, Dy	*	*	*	*	*	
		○		<ea>	*	*	*	*	*	1ビットのみシフト
ASR	○	○	○	Dx, Dy	*	*	*	*	*	
	○	○	○	#<data>, Dy	*	*	*	*	*	
		○		<ea>	*	*	*	*	*	1ビットのみシフト
LSL	○	○	○	Dx, Dy	*	*	*	0	*	
	○	○	○	#<data>, Dy	*	*	*	0	*	
		○		<ea>	*	*	*	0	*	1ビットのみシフト
LSR	○	○	○	Dx, Dy	*	*	*	0	*	
	○	○	○	#<data>, Dy	*	*	*	0	*	
		○		<ea>	*	*	*	0	*	1ビットのみシフト
ROL	○	○	○	Dx, Dy	−	*	*	0	*	
	○	○	○	#<data>, Dy	−	*	*	0	*	
		○		<ea>	−	*	*	0	*	1ビットのみローテート
ROR	○	○	○	Dx, Dy	−	*	*	0	*	
	○	○	○	#<data>, Dy	−	*	*	0	*	
		○		<ea>	−	*	*	0	*	1ビットのみローテート
ROXL	○	○	○	Dx, Dy	*	*	*	0	*	
	○	○	○	#<data>, Dy	*	*	*	0	*	
		○		<ea>	*	*	*	0	*	1ビットのみローテート
ROXR	○	○	○	Dx, Dy	*	*	*	0	*	
	○	○	○	#<data>, Dy	*	*	*	0	*	
		○		<ea>	*	*	*	0	*	1ビットのみローテート

ⅳ) シフトおよびローテート命令

表3.20に，シフトおよびローテート命令のオペランドのフォーマットを示す．各命令とも，メモリ・シフトを指定した場合は，サイズはワードで1ビットのみのシフト（ローテート）に限られるので注意が必要である．

ⅴ) ビット処理命令

表3.21に，ビット処理命令のオペランドのフォーマットを示す．ビット処理の対象がデータ・レジスタのときはレジスタ全体（ロング・ワード）が，メモリのときはバイトが処理の対象となる．

表 3.21 ビット処理命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
BCHG	○		○	Dn, <ea>	—	—	*	—	—	<ea>がDnのときはサイズはロング・ワード，<ea>がメモリのときはサイズはバイトに限られる
	○		○	#<data>, <ea>	—	—	*	—	—	
BCLR	○		○	Dn, <ea>	—	—	*	—	—	
	○		○	#<data>, <ea>	—	—	*	—	—	
BSET	○		○	Dn, <ea>	—	—	*	—	—	
	○		○	#<data>, <ea>	—	—	*	—	—	
BTST	○		○	Dn, <ea>	—	—	*	—	—	
	○		○	#<data>, <ea>	—	—	*	—	—	

ⅵ) 2進化10進数（BCD）処理命令

表3.22に，BCD処理命令のオペランドのフォーマットを示す．BCD処理命令では，コンディション・コードのXビットが演算に使われるので，Xビットの初期状態には注意が必要である．

表 3.22 BCD処理命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
ABCD	○			Dy, Dx	*	U	*	U	*	
	○			−(Ay), −(Ax)	*	U	*	U	*	
NBCD	○			<ea>	*	U	*	U	*	
SBCD	○			Dy, Dx	*	U	*	U	*	
	○			−(Ay), −(Ax)	*	U	*	U	*	

vii）プログラム制御命令

表3.23に，プログラム制御命令のオペランドのフォーマットを示す．

表 3.23　プログラム制御命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
Bcc	○	○		<label>	—	—	—	—	—	<displacement>は符号付き 8，16ビット
	○	○		*±<displacement>	—	—	—	—	—	
DBcc		○		Dn, <label>	—	—	—	—	—	<displacement>は符号付き 16ビット
		○		Dn, *±<displacement>	—	—	—	—	—	
Scc	○			<ea>	—	—	—	—	—	
BRA	○	○		<label>	—	—	—	—	—	<displacement>は符号付き 8，16ビット
	○	○		*±<displacement>	—	—	—	—	—	
BSR	○	○		<label>	—	—	—	—	—	<displacement>は符号付き 8，16ビット
	○	○		*±<displacement>	—	—	—	—	—	
JMP	—	—	—	<ea>	—	—	—	—	—	
JSR	—	—	—	<ea>	—	—	—	—	—	
RTR	—	—	—		*	*	*	*	*	
RTS	—	—	—		—	—	—	—	—	

Bcc, BRA, BSR命令では，オペレーション・コードの後に“.S”を付記することによって，命令を強制的に1ワード長にすることができる（ただし，ディスプレースメントの範囲が－128～＋127を超えた場合はエラーとなる）．しかし，“.S”を付けた命令は，自分自身のすぐ後に続く命令に分岐することはできない．これは，ディスプレースメントの値が0のときは特別な意味（命令長が2ワードを意味する）に使われるからで，注意が必要である．

【例1】

```
          BEQ.S     LABEL     次の命令へジャンプ
                              (アセンブル・エラーとなる)
LABEL     MOVEQ.L   #1, D0
```

viii）システム制御命令

表3.24に，システム制御命令のオペランドのフォーマットを示す．

3.4.4　アセンブラ制御命令

表3.25に，アセンブラ制御命令 (directive) を示す．クロス・マクロ・アセンブラとレジデント・マクロ・アセンブラで使える命令，使えない命令がある

表 3.24 システム制御命令のオペランドのフォーマット

命令	サイズ			オペランドのフォーマット	コンディション・コード					備考
	B	W	L		X	N	Z	V	C	
RESET	−	−	−		−	−	−	−	−	
RTE	−	−	−		＊	＊	＊	＊	＊	
STOP	−	−	−	＃＜data＞	＊	＊	＊	＊	＊	
MOVE			○	USP, An	−	−	−	−	−	
			○	An, USP	−	−	−	−	−	特権命令
		○		＜ea＞, SR	＊	＊	＊	＊	＊	
ANDI		○		＃＜data＞, SR	−	＊	＊	0	0	
ORI		○		＃＜data＞, SR	−	＊	＊	0	0	
EORI		○		＃＜data＞, SR	−	＊	＊	0	0	
TRAP	−	−	−	＃＜vector＞	−	−	−	−	−	
TRAPV	−	−	−	・	−	−	−	−	−	トラップ発生命令
CHK		○		＜ea＞, Dn	−	＊	U	U	U	
MOVE		○		＜ea＞, CCR	＊	＊	＊	＊	＊	
		○		SR, ＜ea＞	−	−	−	−	−	
ANDI	○			＃＜data＞, CCR	−	＊	＊	0	0	ステータス・レジスタ用命令
ORI	○			＃＜data＞, CCR	−	＊	＊	0	0	
EORI	○			＃＜data＞, CCR	−	＊	＊	0	0	

ので注意が必要である．

表3.25で使用している記号の意味は次の通りである．

- △：1個以上のスペースを表わす．
- [ ]：省略可能なことを表わす．
- モジュール名，バージョン，リビジョン：プログラムのオブジェクト・モジュールを識別するための情報
- レジスタ・リスト：MOVE M 命令で扱うレジスタ群
- アドレス・レジスタ：ベース・レジスタとなるアドレス・レジスタ A0～A7
- セクション：セクション番号 0～15

3.4.5 リロケータブルなプログラミング

大規模なプログラムを開発する場合，各機能ごとにプログラムを分割して作成したり，以前開発したプログラムの一部をそのまま活用したりすることがある．このような場合，始めからプログラムをアセンブルし直すのではなく，個

表 3.25 アセンブラ制御命令

命令	分類	機能	フォーマット	サポート言語		備考
				クロス	レジデント	
COMM	プログラム制御	共通データ・エリアの宣言	〈ラベル〉△COMM	○		
DATA	プログラム制御	ローカル・データ・エリアの宣言	〈ラベル〉△DATA	○		
DC	データ定義	定数の定義	[〈ラベル〉]△DC[.サイズ] △〈式〉[.〈式〉……]	○	○	サイズはB, W, L
DCB	データ定義	定数ブロックの定義	[〈ラベル〉]△DCB[.サイズ] △〈個数〉,〈式〉		○	〈個数〉は確保するデータ数
DS	領域確保	メモリの領域確保	[〈ラベル〉]△DS[.サイズ] △〈式〉	○	○	サイズはB, W, L
END	プログラム制御	ソース・プログラムの終り	△END	○	○	
ENDC	条件付きアセンブリ	条件付きアセンブリの終了	△ENDC	○	○	
ENDM	マクロ制御	マクロ定義の終了	△ENDM	○	○	
EQU	値設定	シンボルに値を与える	〈ラベル〉△EQU△〈式〉	○	○	
FAIL	プログラム制御	アセンブル・エラーを発生させる	△FAIL△〈式〉	○	○	〈式〉はエラー番号
FORMAT	リスティング制御	リストのフォーマッティング	△FORMAT	○	○	
IDNT	リンケージ制御	モジュール名の設定	〈モジュール名〉△IDNT△ バージョン，リビジョン		○	
IFEQ	条件付きアセンブリ	〈式〉=0のときアセンブル	△IFEQ△〈式〉	○	○	
IFNE	条件付きアセンブリ	〈式〉≠0のときアセンブル	△IFNE△〈式〉	○	○	
LC	リスティング制御	1ページ当たりのライン数を指定	△LC△〈式〉	○		
LIST	リスティング制御	アセンブル・リストの出力要求	△LIST	○	○	
LLEN	リスティング制御	1行当たりの出力文字数を指定	△LLEN△〈式〉	○	○	
MACRO	マクロ定義	マクロ定義の開始	〈ラベル〉△MACRO	○	○	
MEXIT	マクロ定義	マクロ展開の途中終了	△MEXIT	○	○	
NAM	リンケージ制御	モジュール名の設定	△NAM△〈モジュール名〉	○		
NOFORMAT	リスティング制御	リストのフォーマッティングを抑止	△NOFORMAT	○	○	
NOLIST	リスティング制御	アセンブル・リストの出力抑止	△NOLIST	○	○	
NOMAC	リスティング制御	マクロ展開行の印刷を抑止	△NOMAC	○		

(表3.25の続き)

命令	分類	機能	フォーマット	サポート言語 クロス	サポート言語 レジデント	備考
NOOBJ	オブジェクト出力	オブジェクトの出力を抑止	△NOOBJ	○	○	
NOPAGE	リスティング制御	改ページ抑止	△NOPAGE	○	○	
OPT	オプション指定	アセンブラ・オプションの指定	△OPT△〈オプション〉[,〈オプション〉……]		○	
ORG	プログラム制御	アブソリュート・セクションの宣言	△ORG[.S]△〈式〉	○	○	〈式〉は先頭アドレス
PAGE	リスティング制御	改ページ要求	△PAGE	○	○	
PROG	プログラム制御	プログラム部の宣言	△PROG	○		
REG	プログラム制御	MOVEM用レジスタ・リストの宣言	〈ラベル〉△REG△〈レジスタ・リスト〉		○	
SCT	プログラム制御	リロケータブル・セクションの宣言	△SCT[.S]△〈式〉	○		〈式〉はセクション番号
SECTION	プログラム制御	リロケータブル・セクションの宣言	[〈ラベル〉]△SECTION[.S]△〈式〉		○	〈式〉はセクション番号
SET	値設定	シンボルの値を(再)定義する	〈ラベル〉△SET△〈式〉	○	○	
SPC	リスティング制御	空白行の出力	△SPC△〈式〉	○	○	
TTL	リスティング制御	タイトルの指定	△TTL△〈文字列〉	○	○	
USING	プログラム制御	ベース・レジスタの設定	△USING△〈ラベル〉,〈アドレス・レジスタ〉	○	○	
COMLINE	プログラム制御	コマンド・ライン・エリアの確保	[〈ラベル〉]△COMLINE△〈式〉		○	
IFGE	条件付きアセンブリ	〈式〉≧0のときアセンブル	△IFGE△〈式〉		○	
IFGT	条件付きアセンブリ	〈式〉>0のときアセンブル	△IFGT△〈式〉		○	
IFLE	条件付きアセンブリ	〈式〉≦0のときアセンブル	△IFLE△〈式〉		○	
IFLT	条件付きアセンブリ	〈式〉<0のときアセンブル	△IFLT△〈式〉		○	
XDEF	リンケージ制御	外部参照名の定義	△XDEF△〈ラベル〉[,〈ラベル〉,……]	○	○	
XREF	リンケージ制御	外部参照名の宣言	△XREF[.S]△[〈セクション〉:]〈ラベル〉[,……]	○	○	セクションはセクション番号
OFFSET	領域確保	領域の名称のみを宣言	△OFFSET△〈式〉		○	

個のプログラムのアセンブルの結果であるオブジェクト・モジュール (object module) を，直接結合できれば便利である．このオブジェクト・モジュールの結合を行なうプログラムがリンケージ・エディタ (linkage editor) である．

（1） セクション

i） リロケータブルなセクション

ソース・プログラムをコーディングしている段階では，まだメモリ上のどの領域へロードするのかが決まっていないセクションを，リロケータブル（再配置可能）なセクション (relocatable section) という．アセンブラでは，16個までのリロケータブルなセクションを定義することができる．それらのセクションには0～15の番号が与えられ，別々にアセンブルしたオブジェクト・モジュールの同じ番号のリロケータブルなセクションは，リンケージ・エディタによって一つの連続したセクションにまとめられる．これにより，別々にコーディングし，別々にアセンブルしたプログラム同士でも，実行命令の領域や変数の領域を同じ番号のセクションとしてコーディングしておけば，リンケージ・エディタによって，実行命令の領域，変数の領域としてそれぞれ一つにまとめることができる．

番号の異なる各セクションには，それぞれ独立したロケーション・カウンタ (location counter) が割り当てられる．すなわち，プログラムの途中で番号の違うセクションが定義されると，ロケーション・カウンタは0となり，新しく定義されたセクションのロケーション・カウンタにリセットされる．そして，以前定義していたセクションが再び定義されると，ロケーション・カウンタの値は，以前の定義部分の続きの値にセットされる．

ロケーション・カウンタの値は，各セクションの先頭からの相対アドレスで，実際にロードされるメモリ上のアドレスとは無関係な値である．

ii） アブソリュートなセクション

リロケータブルなセクションが，オブジェクト・モジュールの段階ではまだロードされるメモリ・アドレスの情報をもっていないのに対し，オブジェクト・モジュールの段階でロードされるメモリ・アドレスの情報をもっているセクションを，アブソリュートなセクション (absolute section) という．

アブソリュートなセクションとリロケータブルなセクションのコーディング上の違いは，次の点である．

① セクションの宣言

アブソリュート	ORG 制御命令で行なう．
リロケータブル	SCT（クロス）または SECTION（レジデント）制御命令で行なう．

② 共通データ・エリア

アブソリュート	定義できない．
リロケータブル	定義できる

アブソリュートなセクションの個数は，一つのプログラム内では239以下(正確には，共通データ・エリアと外部参照名の個数を含めて239以下）でなければならない．

アブソリュートなセクションのロケーション・カウンタの値は，ロードされるメモリ・アドレスそのものを指している．

iii） 共通データ・エリア

複数のプログラムから共通に使用されるデータ，変数は，共通データ・エリア内に確保する．共通データ・エリアは，それを参照，更新するプログラム中にエリアが確保されるのではなく，リンケージ・エディタでプログラムの結合が行なわれるときに，一つの領域が共通データ・エリアとして割り当てられる．

（2） 外部参照

いくつかの別々にアセンブルしたプログラムを一つに結合して動作させる場合，あるプログラムから他のプログラム内にあるラベルを参照しなければならない場合がある．このように，他のプログラム内のラベルを参照することを外部参照 (external reference) といい，他プログラムから参照されるラベルを外部定義・シンボル (external symbol) という．

外部参照しているオペランドのアドレス決定は，リンケージ・エディタがプログラムの結合時に行なうが，これに必要な情報は，アセンブラが作成してリンケージ・エディタに渡さなければならない．アセンブラは，XDEF，XREF 制御命令の外部参照定義名の宣言に従って，リンケージ・エディタに渡す情報を作成する．

3.4.6 プログラム例

以下では，アセンブラのプログラム例をいくつか示す．

【例1】　データの転送

```
*          DATA MOVE
*
*MOVE A BLOCK OF DATA FROM SOURCE TO DESTINATION.
*D0 CONTAINS THE NUMBER OF LONG WORDS-1 TO BE MOVED.
*A0 CONTAINS THE SOURCE ADDRESS AND A1 CONTAINS THE
*DESTINATION ADDRESS.
*
           ORG       $1000
AGAIN      MOVE.L    (A0)+, (A1)+    ロング・ワード・データの転送，アドレ
                                     スは自動的に更新される
           DBRA      D0, AGAIN       (D0)＝−1ならば転送終了
           END
```

D0には，転送するロング・ワード・データ数−1をあらかじめセットしておく．この場合，そうすればロング・ワードで所要の回数だけ転送してくれる．

転送するデータのサイズをB，Wとすれば，バイト・データ，ワード・データの転送が可能になる．D0に転送するワード・データ数がセットされている場合のプログラムは

```
           ORG       $2000
AGAIN      MOVE.W    (A0)+, (A1)+    ワード・データの転送
           SUBQ.W    #1, D0          データ数の更新
           BNE.S     AGAIN           (D0)≠0ならば転送継続
           END
```

となる．

【例2】　データの取出し

```
*              MULTIPLE TABLE LOOK UP
*
*USING LOOK UP TABLES, THE ASCII AND DECIMAL CODE VALUES
*OF A HEX DIGIT IN D0 ARE STORED AT LOCATIONS TO WHICH
*A1 IS POINTING.
*A0 POINTS TO THE SET OF LOOKUP TABLES.
*
*
               ORG        $2000
MLOOKUP        EQU        *
               MOVE.L     A0, -(SP)           A0の退避
               LEA        ASCTABLE, A0        テーブルの先頭アドレスをロー
                                              ド
               MOVE.B     0(A0, D0), (A1)+    ASCIIコードをセット
```

```
          MOVE.B    $10(A0, D0), (A1)+     DECIMALをセット
          MOVE.L    (SP)+, A0              A0の回復
          RTS
ASCTABLE  DC.B      $30, $31, $32, $33, $34, $35, $36, $37,
                    $38, $39
          DC.B      $41, $42, $43, $44, $45, $46,
BCDTABLE  DC.B      0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $10, $11, $12,
                    $13, $14, $15
          END
```

D0に格納されたバイト・データに対応するASCIIコードとBCDを取り出すサブルーチンである.

【例3】 文字列の比較

```
*         STRING COMPARE
*
*TWO STRINGS ARE COMPARED FOR SAMENESS.
*A0 POINTS TO THE BEGINING OF THE FIRST STRING AND A1, TO
*THE BEGINING OF THE SECOND.
*THE NUMBER OF LONG WORDS TO BE COMPARED IS IN D1.
*
          ORG       $2000
          BRA.S     SKIP              比較する文字数が0の場合のチェック
AGAIN     CMPM.L    (A0)+, (A1)+      文字列（ロング・ワード）の比較
SKIP      DBNE      D1, AGAIN         不一致および（D1）キ−1ならば比較
                                      の継続
NOCMP     NOP
          END
```

【例4】 32ビット符号付き乗算

```
*         32*32 BIT SIGNED MULTIPLY
*
*MULTIPLY TWO 32 BIT SIGNED VALUES TO GENERATE A 64 BIT
*SIGNED PRODUCT.
*UPON ENTERING THE SUBROUTINE, SV1 IS IN D0 AND SV2 IS IN
*D1.
*UPON EXITING THE SUBROUTINE, THE LOWER 32 BITS OF THE
*RESULT ARE IN D1 AND THE UPPER 31 BITS PLUS SIGN BIT ARE
*IN D0.
*
          SCT       1
          XDEF      DMULTS            外部定義名の宣言
DMULTS    EQU       *
          MOVEM.L   D2-D6, -(SP)      レジスタD2〜D6の退避
```

```
        CLR.B     D6              符号記憶インディケータのリセット
        TST.L     D0              符号のチェック
        BPL.S     CHKD2
        NEG.L     D0              符号の反転(－→＋)
        NOT.B     D6              インディケータのセット
CHKD2   TST.L     D1              符号のチェック
        BPL.S     READY
        NEG.L     D1              符号の反転(－→＋)
        NOT.B     D6              インディケータのセット
READY   EQU       *
        MOVE.W    D0, D3          下位ワード・データをD3へロード
        SWAP      D0
        MOVE.W    D0, D2          上位ワード・データをD2へロード
        MOVE.W    D1, D5          下位ワード・データをD5へロード
        SWAP      D1
        MOVE.W    D1, D4          上位ワード・データをD4へロード
        SWAP      D1
        MULU      D4, D0          (上位ワード・データ)×
                                    (上位ワード・データ)
        MULU      D3, D1          (下位ワード・データ)×
                                    (下位ワード・データ)
        MULU      D5, D2          (下位ワード・データ)×
                                    (上位ワード・データ)
        MULU      D4, D3          (上位ワード・データ)×
                                    (下位ワード・データ)
        SWAP      D2
        SWAP      D3
        CLR.L     D4
        CLR.L     D5
        MOVE.W    D2, D4
        MOVE.W    D3, D5
        CLR.W     D2
        CLR.W     D3
        ADD.L     D2, D1        ┐
        ADDX.L    D4, D0        │
        ADD.L     D3, D1        ├ 積の計算
        ADDX.L    D5, D0        ┘
*
        TST.B     D6              積の符号のチェック
        BEQ.S     DMEND
        NEG.L     D1            ┐ 符号の反転(＋→－)
        NEGX.L    D0            ┘
DMEND   EQU       *
*
        MOVEM.L   (SP)+, D2-D6    レジスタD2～D6の回復
        RTS
        END
```

結果は，D0に上位，D1に下位が入り，D0の最上位ビットが符号となる．

【例5】 データの整列

```
*          SEQUENCER
*
*SEQUENCES A STRING OF WORD DATA SUCH THAT THE LARGEST
*NUMBER IS IN THE LOWEST MEMORY LOCATION.
*A0 POINTS TO THE BEGINNING OF THE STRING, AND A1, TO THE
*END.
*
*
           SCT       2
           XDEF      SEQUENCR              外部定義名の宣言
SEQUENCR   EQU       *
           MOVEM.L   A0/A2/D0, -(SP)       レジスタの退避
           MOVEA.L   A0, A2                先頭アドレスの退避
BGNAGN     MOVEA.L   A2, A0                比較アドレスのイニシャル
NXTPR      CMPM.W    (A0)+, (A0)+          ワード・データの比較
           BGT.S     EXCHNG                上位アドレスのデータのほうが
                                           大きいときは，データを入れ
                                           換える
           TST       -(A0)                 データ・アドレスの修正(-2)
           CMPA.L    A0, A1                比較の終りをチェック
           BNE.S     NXTPR
           MOVEM.L   (SP)+, D0/A0/A2       レジスタの回復
           RTS
EXCHNG     EQU       *
           MOVE.W    -(A0), D0             ┐
           MOVE.W    -(A0), 2(A0)          ├データの入換え
           MOVE.W    D0, (A0)              ┘
           BRA.S     BGNAGN
           END
```

符号付きワード・データを，値の大きいものから順番に並べ換えるプログラムである．A0にデータの先頭アドレス，A1にエンド・アドレスがセットされている．

【例6】 データの挿入

```
*        ADD TO A SEQUENCED LIST
*
*A NEW NUMBER IS INSERTED INTO THE PROPER PLACE OF A
*SEQUENCED LIST.
*THE NUMBER TO BE INSERTED IS IN D0. THE LOWEST ADDRESS
*OF THE LIST IS IN A0. THE HIGHEST ADDRESS+1 IS IN A1.
```

```
*THE LARGEST DATA IS IN THE LOWEST ADDRESS, AND DATA IS
*MOVED DOWN TO MAKE ROOM FOR THE NEW WORD.
*
        ORG     $2000
        CMP     (A0), D0        最大値と比較
        BGE.S   FINAL           最大値より大きいときはブランチ
MVDWN   MOVE    (A0)+, -4(A0)   データを移動して空きを作る
        CMP     (A0), D0        データを比較
        BGE.S   FINAL           大きいときはブランチ
        CMP     A0, A1          終りのチェック
        BHI     MVDWN
FINAL   MOVE    D0, -(A0)       新しいデータをセット
        RTS
        END
```

値の大きいものから順に並んでいるデータの列に，新しいデータを追加，挿入するプログラムである．追加によりはみ出たデータは，アドレスの小さい方に順番にずれていく．

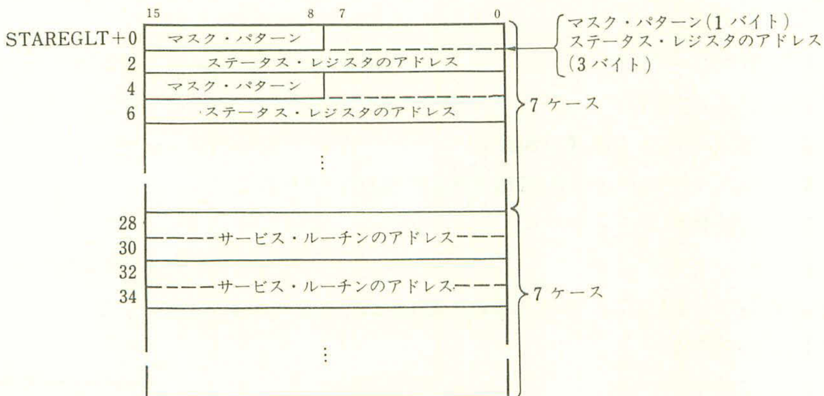
【例7】 CRTC イニシャライズ

```
*       CRTC INITIALIZATION PROGRAM
*
*THIS ROUTINE CAN BE USED TO INITIALIZE THE REGISTERS OF
*THE CRT CONTROLLER IN AN 68000 SYSTEM.
*
*
        ORG     $9000
CRTC    DS.B    1               CRTCポインタ・レジスタ
        ORG     $9002
        DS.B    1               CRTCポインタ・レジスタ
        ORG     $3000
        LEA     CRTC, A1        ポインタ・レジスタのアドレスをロード
        LEA     TABLE, A0       初期値テーブルのアドレスをロード
        CLR     D0
AGAIN   MOVE.B  (A0)+, D0       初期値をロード
        MOVEP   D0, 0(A1)       初期値をセット
        ADD     #$100, D0       ポインタの値を更新
        BTST    #12, D0         終りをチェック(16回ループ)
        BEQ.S   AGAIN
        ―――
        ―――
        ORG     $31F0
TABLE   DC.B    $65, $50 ,$56, $09, $18, $0A, $18, $18
        DC.B    $00, $0B, $00, $0B, $00, $80, $00, $80
        END
```

【例8】 ポーリング・ルーチン

```
*               PERIPHERAL POLLING ROUTINE
*
*THIS IS A POLLING ROUTINE FOR A SYSTEM WHICH CONTAINS
*6800 PERIPHERAL DEVICES.
*THE TABLE, STAREGLT, CONTAINS THE ADDRESS OF EACH
*PERIPHERAL STATUS REGISTER AND A STATUS MASK. ANOTHER
*TABLE, LOCATED 'LENGTH' FROM STAREGLT, CONTAINS THE
*ADDRESS OF EACH SERVICE ROUTINE.
*
                ORG        $5000
                LEA        STAREGLT, A0          テーブルの先頭アドレスをロード
                MOVEQ.L    #-4, D1               インデックスのイニシャライズ
AGAIN           ADDQ.W     #4, D1                インデックスの更新
                CHK        #LENGTH, D1           (D1)>LENGTHならば
                                                 TRAP割込み発生
                MOVE.L     0(A0, D1), D0         ステータス・レジスタの
                AND.L      #$FFFFFF, D0          アドレスを取り出す
                MOVEA.L    D0, A1
                MOVE.B     (A1), D0              ステータス・レジスタをread
                AND.B      0(A0, D1), D0         ステータス・ビットをマスク
                BEQ.S      AGAIN
                MOVE.L     LENGTH(A0, D1), A0    対応するサービス・ルーチ
                                                 ンのアドレスをロード
                JMP        (A0)                  サービス・ルーチンへジャンプ
STAREGLT        DS.L       14
LENGTH          EQU        28
                END
```

下記のSTAREGLTテーブルに従って，周辺装置のポーリングを行なうプ



ログラムである.

3.5 高級プログラミング言語

68000で使える高級プログラミング言語としては, FORTRAN, Pascal, BASIC, C, COBOLなどの一般的な言語のほかに, 68000に固有な言語としてS-PL/Hがある. 以下では, (株) 日立製作所から提供されている高級言語の中から, S-PL/HとFORTRANについて簡単にその特徴などを記す.

3.5.1 S-PL/H

S-PL/H (super programming language of Hitachi micro computer) は8086用の高級言語PL/M-86に相当する言語で, CP/M流の呼び方にならえばPL/M-68000とも呼べる言語である.

S-PL/Hは汎用高級言語PL/Iからマイクロコンピュータの応用システム記述に必要な機能を抜き出し, さらにマイクロコンピュータに特有の機能を付加して構成した高級プログラミング言語である. S-PL/Hのプログラムはアセンブラに比べ読みやすく, また少ないステップ数で記述できるので, S-PL/Hを使用すると, プログラムの生産性と品質を向上することができる.

S-PL/Hは, 8ビットの6800用高級言語PL/Hと上方への互換性 (upward compatible) があり, PL/M-86とはハードウェアに密着したわずかの部分を除いて互換性がある.

S-PL/HがPL/HおよびPL/M-86より機能的に拡張されている項目は, 次の通りである.

(1) 3レベル構造体

(2) 3次元配列

(3) 選択子付き DO CASE文

(4) マルチプロセッサ制御用TEST AND SET文

(5) 倍長整数型データ

(6) ビット型データ

上記以外に, PL/Hにプラスされた機能は次の通りである.

(1) 整数型データ

(2) 実数型データ

(3) 手続きのリエントラント属性

S-PL/Hのコンパイラには，クロス方式とレジデント方式の2種類がある．

3.5.2 FORTRAN

FORTRAN (FORmula TRANslation) は科学技術計算用として広く使われている高級プログラミング言語である．68000のFORTRANは，ANSI FORTRAN 77 (american national standard programing language FORTRAN) のサブセットFORTRANの規格を満たし，さらにISA-S61.1 (industrial computer system FORTRAN procedures for executive functions, process input/output, and bit manipulation) の中のビット処理機能を追加したものである．

68000のFORTRANには，次の特徴がある．

(1) ブロックIF文が使えるので，余計な文番号を書かずに制御構造が記述でき，構造化プログラミングが容易に可能になる．

(2) 文字型の変数，配列要素および配列が使用できる．

(3) メモリ内のデータを内部ファイルとして取り扱うことができるので，データ編集が容易に行なえる．

(4) 副プログラムの変数および共通ブロックの内容を，SAVE文によってRETURN文またはEND文の実行後でも保持することができる．

FORTRAN77のサブセットに追加されている68000 FORTRANの機能は，次の通りである．

(1) 整数型データとして4バイトのほかに2バイト・データが使用できる．

(2) 実数型データとして4バイトのほかに8バイト・データが使用できる．

(3) 上記(1)および(2)の追加に伴い，型宣言文にデータの長さ指定が追加されている．

(4) 次の10種類の組込み関数および組込み手続きが追加されている．

IOR (論理和)，IAND (論理積)，NOT (論理否定)，IEOR (排他的論理和)，ISHFT (シフト)，IBSET (ビット・セット)，IBCLR (ビット・クリア)，BTEST (ビット・テスト)，INPUT (絶対アドレス入力)，OUTPUT (絶対アドレス出力)．

(5) 次の組込み関数が，引数に使用できないものとして追加されている．

IOR, IAND, NOT, IEOR, ISHFT, IBSET, IBCLR, BTEST

（6） BLOCK DATA文が使用できる.

（7） 16進定数，nH型文字定数が使用できる.

（8） SAVE文に共通ブロック名のほかに，変数名，配列名を書ける.

3.6 デ バ ッ ガ

3.6.1 デバッガの概要

68000デバッガは，EPROMのファームウェアとして実装され，H680SBCシステムを使用するユーザ・プログラムの開発支援のために用意されたデバッグ・ツールである.

68000デバッガは，オブジェクト・プログラムの入出力，I/O装置の割付け，レジスタ・メモリ内容の表示・変更など豊富な機能をもっている．また，68000アセンブラおよびテキスト・エディタも68000デバッガの下で動作する.

デバッガはオブジェクト・テープのロード，ベリファイ，メモリ内容のプリント，パンチ，転送，入出力機器の選択，アセンブラ，テキスト・エディタへの制御の移動，ユーザ・プログラムのデバッグなどの機能をもっている.

ユーザはデバッガの上記機能を使用して，プログラム開発を効率よくできるようになっている.

3.6.2 デバッガの機能

デバッガは以下のような機能をもっている.

（1） プログラムのロードとパンチ

i） メモリへ紙テープのSタイプ・オブジェクト†をロードする(LO機能).

ii） メモリの内容と紙テープ・オブジェクトの内容を照合する(VE機能).

iii） メモリ内容をSタイプ・オブジェクト形式で紙テープに打ち出す(PU機能).

（2） メモリ・レジスタ内容の表示，変更

i） メモリ内容の参照および変更ができる(MD・MM機能).

ii） レジスタ内容の参照および変更ができる(DF・RM機能).

iii） メモリ間でメモリ内容の転送ができる(MV機能).

---

† Sタイプオブジェクトのフォーマットについてはこの章の最後に説明する.

ⅳ）指定アドレス間のメモリ内容をクリアする（BT機能）.

（3）プログラムの実行

ⅰ）指定アドレスからのプログラム実行ができる（GO機能）.

ⅱ）ユーザ・プログラムのトレースができる（TR機能）.

ⅲ）ブレーク・ポイントの設定，追加，取消し，表示ができる（BR機能）.

ⅳ）ユーザ作成のサブルーチンを実行する（CA機能）.

（4）エクセプション処理，入出力関係

ⅰ）バス・エラー，アドレス・エラーについて，ユーザ・プログラムで使用するベクタ・アドレスの設定ができる（VA機能）.

ⅱ）入出力ルーチンの選択（I/Oルーチンの割付け）ができる（IO機能）.

ⅲ）プリンタの接続ができる（PA機能）.

ⅳ）ACIAによるデータ転送におけるワード構成の変更および文字間のnull数の変更ができる（PF機能）.

（5）その他

ⅰ）アセンブラへ制御を移すことができる（AS機能）.

ⅱ）テキスト・エディタへ制御を移すことができる（TX機能）.

ⅲ）データの変換を行なう（DC機能）.

3.6.3 デバッガの標準的な使い方

デバッガを使用したプログラム・デバッグ手順（図3.16）について説明する．まず，テキスト・エディタおよびアセンブラにより作成したオブジェクト・プログラムをSBCシステムのメモリ上にロードする．Sタイプ・オブジェクト・プログラムの形で紙テープからロードする場合はLOコマンドを，EPROMから転送する場合はMVコマンドを使う．

次にデバッグ用コマンドを用い，プログラムをデバッグする．デバッグ用コマンドには，レジスタ内容表示・変更用のDF・RMコマンド，メモリ内容表示・変更用のMD・MMコマンド，ユーザ・プログラム実行用のGO・GTコマンド，トレース用のTR・TTコマンド，ブレーク・ポイント設定用のBRコマンドなどがある．

デバッグしたプログラムはPUコマンドにより，Sタイプ・オブジェクト・プログラムの形で紙テープにパンチされる．

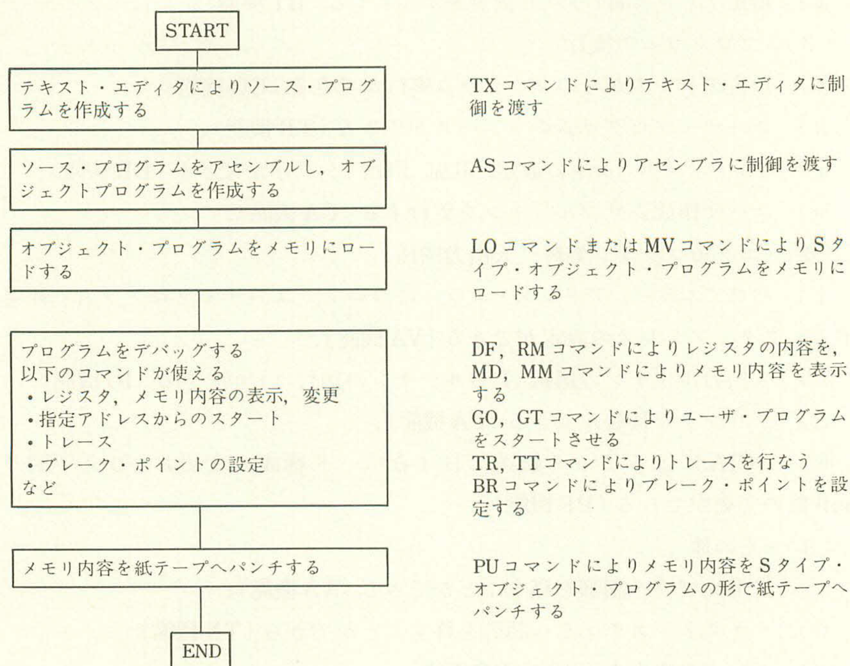


図 3.16 プログラム・デバッグの手順

3.6.4 デバッガ・コマンド

デバッガでは，コマンド待ちの記号としてコンソール出力装置の先頭に“／”を表示する．以下に示すコマンド一覧表に従ってコマンドをコンソールよりキーインする．

デバッガのコマンド・シンタックスは以下のようになっている．

（1） $\begin{Bmatrix} A \\ B \\ C \end{Bmatrix}$ はA，B，Cの内のどれか一つを選択することを示す．

（2） [ ] は省略可能であることを示す．

（3） ( )... は( )内の繰返しであることを意味する．

（4） アンダーラインの部分は，ユーザがキーインすることを意味する．

（5） コマンドの1文字消去

入力中のコマンドの1文字消去は (BS) (back spaceの意，(CTRL)キー

＋H）または (DEL) をキーインする．直前の文字に対しては (BS) を1個，1個前の文字には (BS) を2個キーインしてから訂正すべき文字をキーインする．(BS) の最初のキーインで“＼”(back slash) を表示する．

またその場合 (BS) 1個に対して，対応する文字をエコーとして表示する．入力文字数以上を消去しようとするとデバッガのコマンド入力要求状態になる．最後のエコー文字の次に“＼”を表示する．

【例1】 NOPA とキーインすべきところを NPQ とキーインしてしまった場合

```
／NPQ (BS)  (BS) OPA      ユーザのキーイン手順
／NPQ＼Q    P＼OPA        コンソール上の表示
       エコー
```

（6） コマンドの1行消去

キーインしたコマンドの1行全部消去には (CAN) (cancel の意，(CTRL) キー＋X キー) をキーインする．

【例2】 MV とキーインした後，コマンドを消去する場合

```
／MV  (CAN)   ユーザのキーイン手順
／MV＼        コンソール上の表示
／
```

（7） コマンドのキャンセル

一つのコマンド機能の全部のキャンセルとして (ESC) (escape の意) をキーインする．(ESC) をキーインすると，該当コマンドの処理は中断され，次のコマンドの入力待ちとなる．

【例3】

```
／MV  (CR)
BEG 1000 (ESC)＼
／                     次のコマンドの入力待ち
```

（8） コマンド入力文字の表示

現在までに入力した文字を表示するには，EOT（(CTRL) キー＋D キー）をキーインする．(BS) キーを使用したときに有用である．

【例4】

```
／NPQ (BS) (BS) OS (BS) PA (EOT)    ユーザのキーイン手順
／NPQ＼QP＼OS＼S＼PA               コンソール上の表示
 NOPA
最初にPQが消え，OSが入り，次にSが消え，PAが入る
```

（9） コマンドに続くオプション指定において，存在しないオプションを指定した場合は無視される．

3.6.5 コマンドの例

デバッガ・コマンドの中でDCコマンドとDFコマンドを例にとり，指定の仕方，使用法について説明する．

（1） DCコマンド

DC △ <式> (CR)

i） 機能

① 式を16進と10進に変換する．

② 式は16進，10進のどちらでもよく，両方とも表示できる．

ii） 説明

① <式>は，A，−A，A+B，A−Bの形のみ許されている．A，Bは正の10進数または16進数である．

② 10進数には&，16進数には$を頭に付けて表現する．

iii） 注意事項

① DCコマンドにおいて数値に$や&を省略した場合は，10進数と解釈する．

② 説明 i）で記述した式の形の後に，+または−で値を付けた場合，その値は無視される．

iv） 使用例

①
```
／DC  &255 (CR)
$FF=&255
／
```

10進数の255を16進数に変換すると$FFになる．

②
```
／DC  $10 (CR)
$10=&16
／
```

16進数の＄10を10進数に変換すると16になる．

③　/DC　&20＋&12 (CR)
　　$20＝&32
　　/

10進数の20と12を加えると32で，16進数の$20になる．

④　/DC　$200－$100 (CR)
　　$100＝&256
　　/

16進数の＄200から＄100を引くと＄100で，10進数の256になる．

⑤　/DC　&10－&100 (CR)
　　－$5A＝－&90
　　/

10進数の10から100を引くと－90で，16進数の－＄5Aになる．

⑥　/DC　$10＋&15 (CR)
　　$1F＝&31
　　/

16進数の＄10と10進数の15を加算すると10進数で31，16進数で＄1Fになる．

（2）DFコマンド

DF { [[△，レジスタ・キーワード)…… / [△ALL] } (CR)

ｉ）機能

ブレーク・ポイント，トレース時または割込みが発生したときに表示するレジスタを選択する．

ii）説明

＜レジスタ・キーワード＞とスペースを区切りにして，コマンド・ストリングに並べることによって，指定レジスタをレジスタ情報†に加える．＜レジスタ・キーワード＞には以下のものがある．

PC　　プログラム・カウンタ

† レジスタ情報とは，ブレーク・ポイント，トレース時または割込みが発生したときに表示される各種レジスタの内容表示をいう．

SR　ステータス・レジスタ
US　ユーザ・スタック・ポインタ
SS　スーパバイザ・スタック・ポインタ
A0～A7　8個のアドレス・レジスタ
D0～D7　8個のデータ・レジスタ

ⅲ）注意事項

① 表示していたレジスタを表示しないようにDFコマンドで変更することはできない．そのような場合にはNODFコマンドにより全レジスタを消去した後に，DFコマンドで表示するレジスタを選択する必要がある．

② DF (CR) とキーインすると，このコマンドで指定されているレジスタ情報を表示する．

③ DF△ALL (CR) とキーインすると，すべてのレジスタ情報を指定したことになる（初期状態では，(ALL)指定になっている）．

④ SR（ステータス・レジスタ）内容の詳細を表示するため，下記のように表示する．

SR=××××××××× = { TS I XNZVC / . . I . . . . . }

各フラグがセットされているとき，そのフラグを意味する文字を表示し，リセットされているときは‘．’を表示する．

Iは16進数で割込みマスク0～7を表わす（表3.26）．すなわちI2，I1，I0を1桁の16進数と見て表示している．

各フラグの意味は次ページの図3.17の通りである．

割込みの種類には次の①～⑤のものがある．

表 3.26　割込みマスク

I	I2	I1	I0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

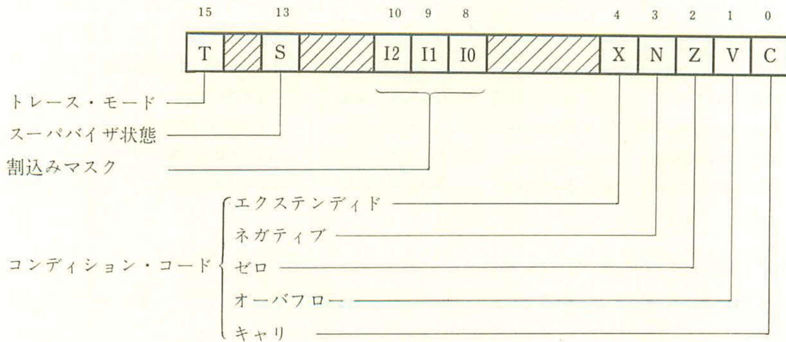


図 3.17

① PIA, ACIA と PTM からの割込み

② アボート・スイッチ

③ リセット・スイッチ

④ バス・エラーの発生

⑤ アドレス・エラーの発生

iv） 使用例

① ／DF (CR)

```
PC=001ED0      SR=2000=.S0.....    US=00FEFF00     SS=00FEFF00
D0=01234567    D1=0000ABCD    D2=00000000    D3=000000D3
D4=00000000    D5=00000000    D6=00000000    D7=00000000
A0=00001000    A1=00002000    A2=0FD21A2C    A3=00000000
A4=00000000    A5=00000000    A6=00000000    A7=00FEFF00
/
```

現在，指定されているレジスタ情報を表示する．

② ／NODF (CR)

```
/DF (CR)
/
```

NODF コマンドですべてのレジスタ情報を解除し，空行を表示する．

③ ／DF.D0 (CR)

```
/DF (CR)
D0=01234567
/
```

レジスタ情報に D0 のみを指定し表示する．

```
④　/DF ALL ⒸⓇ
    /DF ⒸⓇ
    PC=001ED0     SR=2000=.S0.....    US=00FEFF00    SS=00FEFF00
    D0=01234567   D1=0000ABCD         D2=00000000    D3=00000000
    D4=00000000   D5=00000000         D6=00000000    D7=00000000
    A0=00001000   A2=00002000         A2=0FD21A2C    A3=00000000
    A4=00000000   A5=00000000         A6=00000000    A7=00000000
    /
```

さらにALLを指定することにより，全レジスタ情報を表示する．

3.6.6　Sタイプ・オブジェクトのフォーマット

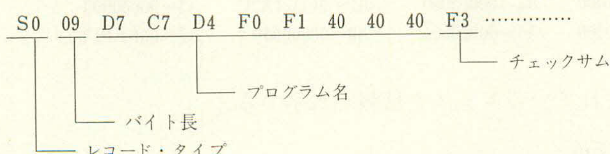
リロケータブル・オブジェクト・モジュールをクロスリンケージ・エディタでまとめると，このSタイプ・ロード・モジュールが出力される．その1レコードは80バイトの長さをもち，ヘッダーS0，データS1，S2，エンドS8，S9に分かれる．その形式は

Sn，バイト長，{プログラム名／ロード・イメージ／エントリ・アドレス}，チェックサム

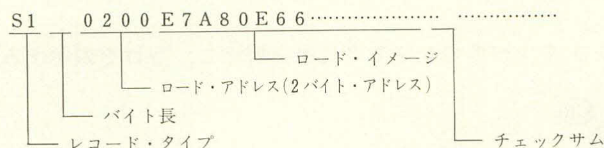
を基本とし，コードはEBCDICかASCIIである．

以下にPGM01というプログラム名の例を示す．これはEBCDIC codeを用いている．

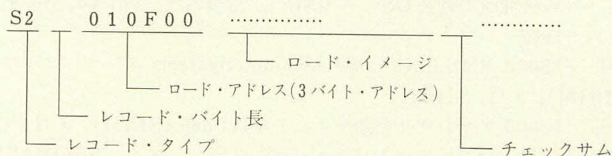
(1)　S0レコード（ヘッダー・レコード）



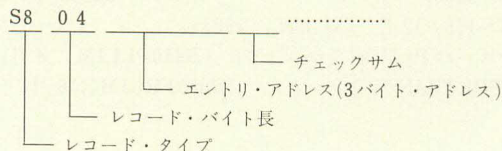
(2)　S1レコード（データ・レコード：ショート・アドレス）



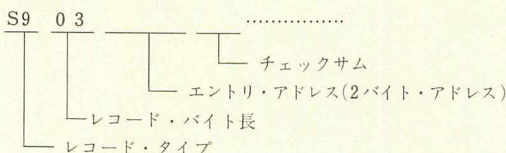
(3) S2 レコード（データ・レコード：スタンダード・アドレス）



(4) S8 レコード(エンド・レコード：エントリ・アドレスが3バイト・アドレスの場合)



(5) S9 レコード(エンド・レコード：エントリ・アドレスが2バイト・アドレスの場合)



ロード・アドレス――このアドレスから次のデータをしまいこむ
エントリ・アドレス――プログラムのスタート・アドレス

参考書

1) 日立製作所：『CP/M-68000 仕様書（暫定版）』(1982).

2) 伊藤　誠：「CP/M ってなんだろう」, インターフェース, No. 50, p. 110～120, 7月, CQ 出版 (1981).

3) D. M. Ritchie, K. Thompson : "The UNIX Timesharing System", *The Bell System Technical*, Vol. 57, No. 6, p. 1947～1969, (1978)：石畑　清, 小野芳彦訳：「UNIX タイムシェアリング・システム」, bit, Vol. 13, No. 9, p. 19～34, 共立出版 (1981).

4) 石田晴久,「UNIX システム入門」, bit, Vol. 13, No. 11, p. 26～32, 以降18回連載, 共立出版 (1981).

5) 石田晴久,「スーパーミニコンやミニコンの共通 OS としての UNIX」, bit, Vol. 13, No. 8, p. 4～9, 共立出版 (1981).

6) 林　秀幸,「プログラミング環境を変える UNIX」, 日経コンピュータ, 4 月5日号,

No. 14, p. 71～87, 日経マグロウヒル (1982).
7) 石田晴久, 「ベル研究所の軽装OS――UNIX」, 情報処理, Vol. 18, No. 9, p. 942～949, 9月 (1977).
8) 日立製作所：『68000 RMS (Real time Monitor System) ユーザーズマニュアル』(S680RMS1M), 2月, (1982).
9) 日立製作所：『68000 アセンブリ言語マニュアル』(S680ASL1M), 3月, (1981).
10) 日立製作所：『68000 マクロアセンブラユーザーズマニュアル』(S680MAS1M), 3月, (1982).
11) 日立製作所：『68000 クロスマクロアセンブラユーザーズマニュアル』, (1982).
12) 荒井正幸, 福留五郎, 「68000 アセンブラ・プログラミングの基礎」, インターフェース, No. 55, p. 134～158, 12月, CQ出版, (1981).
13) 日立製作所：『68000 スーパPL/H言語マニュアル』(S680PLL1M), 8月, (1981).
14) 日立製作所：『68000FORTRAN言語マニュアル』(S680FRL1M), 8月, (1981).

4章 応用例

16ビット・マイクロプロセッサを新規に採用した機器のうち，68000は，1980年には全体の9.9%にすぎなかったが，1981年には41.8%に達した．この伸びはある意味で68000の優秀性を物語っている．ここではその応用例を紹介するが，資料が何とか入手でき，しかも読者が興味をもち，ある程度は身近なものとなると当然パーソナルコンピュータやシステム開発装置などが中心になる．集めた資料は精粗さまざまであり，大変面白いものもあった．著者はこれらの資料に基づき，なるべくわかりやすくまとめるように努力したが，ある場合は詳しく，ある場合は雑になっている．これらを含め，関係各社の御厚意に深く謝意を表する．

68000および周辺LSIを用い，高度なコンピュータ・システムを作ることができるので，こういうシステムがいわゆるミニコンピュータの分野に進出することは当然予想される．特に，68010や68020を中心としたシステムでは仮想記憶も使えるので，大変高級なシステムが完成しよう．これからの進展が大変楽しみである．

4.1 学習用モジュール

アセンブラが使える最小の教育用システムとしてMotorola社のSpeed Master 68K（図4.1，4.2），（株）日立製作所のH680TR01（図4.3）があげられる．どちらも32Kバイトのメモリをもつシングルボード・コンピュータで，オーディオ・カセットやプリンタなどをつないで使うことができる（表4.1）．言語としてはアセンブラだけしかなく，他の高級言語はこのままでは使えない．

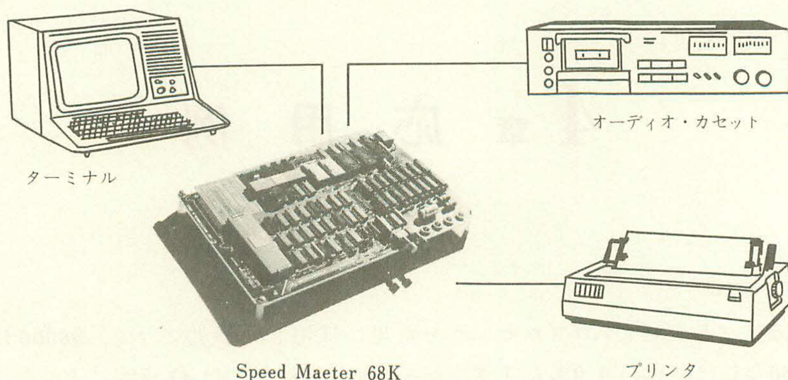


図 4.1 Speed Master 68K (Motorola社の提供)

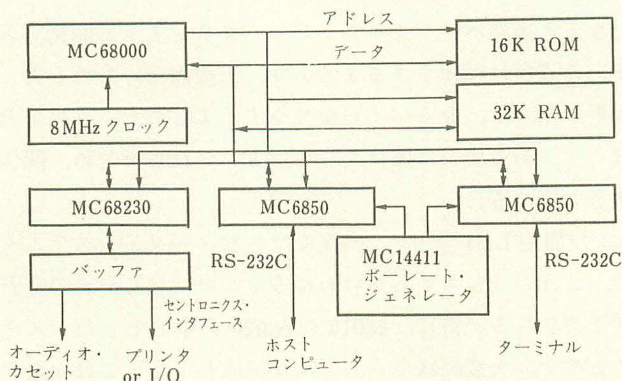


図 4.2 Speed Master 68Kのブロック・ダイアグラム (Motorola社の提供)

表 4.1 学習用モジュール

	Speed Master 68K	H680TR01
主たるマイクロ・プロセッサ	68000 L4 (4 MHz)	68000 L4 (4 MHz)
モニタの大きさ	8 K バイト ROM×2	8 K バイト ROM
I/Oポート	RS-232C インタフェース×2 セントロニクス・インタフェース (プリンタ用) カセット用インタフェース (直結可能)	
ボーレート	可変：110, 150, 300, 600, 1200, 2400, 4800, 9600	
ソフトウェア	アセンブラ, 逆アセンブラ, デバッガ	
メモリ	32 K バイト RAM 最大 128 K バイト (増設可能)	

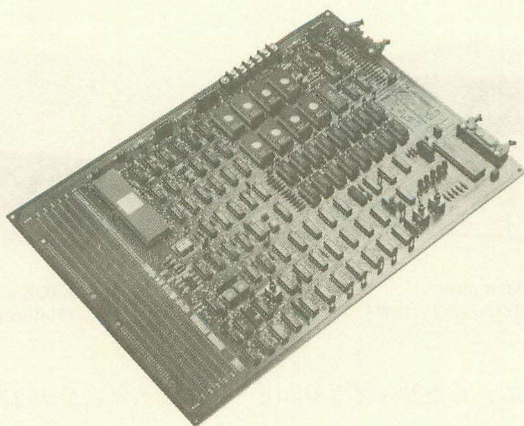


図 4.3 H680TR01 ((株)日立製作所の提供)

4.2 システム開発装置

これは制御などで必要なプログラムを開発するための装置で，Motorola 社では開発支援装置という．プログラムに用いる言語はアセンブラだけではなくコンパイラも含まれており，完成したプログラムのデバッグ，リンク，エミュレーションが必要である．ここでは Motorola 社のEXOR macs，（株）日立製作所の H680SD300，安立電気（株）の μPDS D7800 を解説するが，3社ともそれぞれに標準システムがあり，いろいろな機能がある．原始プログラムから再配置可能なオブジェクト・プログラムが作成され，これらをリンクして使うようになっているのでデバッグは容易である．コンパイラもシステム開発に便利なように作られており，ビット処理ももちろん可能である．

4.2.1 EXOR macs（Motorola 社）

標準システムは 384 K バイトのメモリ，ディスク・コントローラ（フロッピーまたはハード・ディスク），デバッグ・モジュール，MPU module の 6 枚から成る．ただしフロッピー・ディスク・コントローラは 1 枚で 512 K バイト×4 ユニット＝2 M バイトを処理する能力をもち，2 ユニットを標準装備している．ハード・ディスクは 16 M バイトの取りはずし可能なカートリッジ形のものと 16 または 80 M バイトの固定ディスクとから成り，フロッピーもハードももう



図 4.4　EXOR macs
(Motorola 社の提供)



図 4.5　EXOR diskIII
(Motorola 社の提供)

1組を装備できる．したがってフロッピー・ディスクは最大2Mバイト，ハード・ディスクは最大192Mバイトまで拡張できる．

いずれもオペレーティング・システムVERSA dosを塔載しており，マルチユーザ・システムとして使うことができる．12インチのグリーンCRTを標準装備しているが，これは110ボーから9600ボーまでの幅でボーレートを選択でき，画面の大きさは24行×80字である．図4.6はCRTターミナルのEXOR termである．

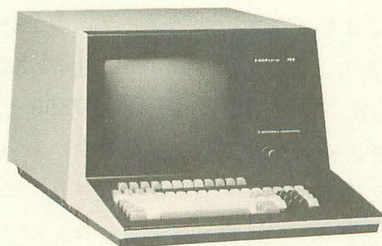


図 4.6　CRTとキーボード：EXOR term
(Motorola 社の提供)

プリンタは180字/秒の速さをもつドット・プリンタで，セントロニクス・インタフェースで接続され，1分間に70行打てる．紙は幅38cmのビジネス・フォームで128字を1行とし，7×7ドットのASCII codeを用いる（図4.7）．

オペレーティング・システムVERSA dosを用いてリアルタイムでソフトウェアを開発できるようになっており，マルチ・タスクでも処理できるので複数の利用者が同時使用できる．主な内容は次の通りである．

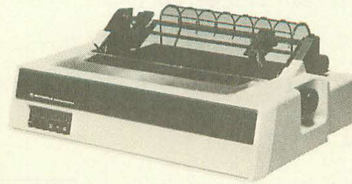


図 4.7　EXOR print
(Motorola 社の提供)

（1） CRT エディタ

（2） ストラクチャード・マクロ・アセンブラ

（3） FORTRAN（オプション）

（4） Pascal

（5） リンケージ・エディタ

（6） シンボリック・デバッガ

（7） その他

このほかオプションでBASIC-M, COBOL, Ada, APL, FORTH などを使うことができる．また言語Cとそれによるオペレーティング・システムUNIXも利用できるようになった．68020を用いたEXOR macsがもう少しで出まわるはずだが，そこではAdaが主流になるということである．

FORTRANやPascalについては第1章で説明したが，前者はANSI FORTRAN 77のサブセットを含み，ISAの拡張を満たすリアルタイム・プロセッシング機能を備えている．FORTRANやPascalは，ソース・プログラムをリロケータブルなオブジェクト・プログラムに翻訳し，それをほかのオブジェクト・プログラムとリンケージ・エディタで結合できるようになっている．

すなわちリンケージ・エディタはコンパイラやアセンブラを用いて作った再配置可能なオブジェクト・プログラムを結合し，ロード・モジュールにまとめる．このとき誤りが発見されればそれを指摘し，モジュール・マップやシンボル・テーブルなどを表示してくれるので，デバッグが容易である．

なお，6800や6809などとのクロス・アセンブラも完備しており，いろいろなマイクロプロセッサと（言語のうえで）コンパチブルである．またコンパイラによるオブジェクト・プログラムは，次に述べるHD680SD300と完全に共通である．

4.2.2 H680SD300（(株) 日立製作所）

これも 256 K バイトのメモリと 2 台 1 組のフロッピー・ディスクを標準装備している．このディスクは 1 台 512 K バイトの容量をもち，全部で 1 M バイト

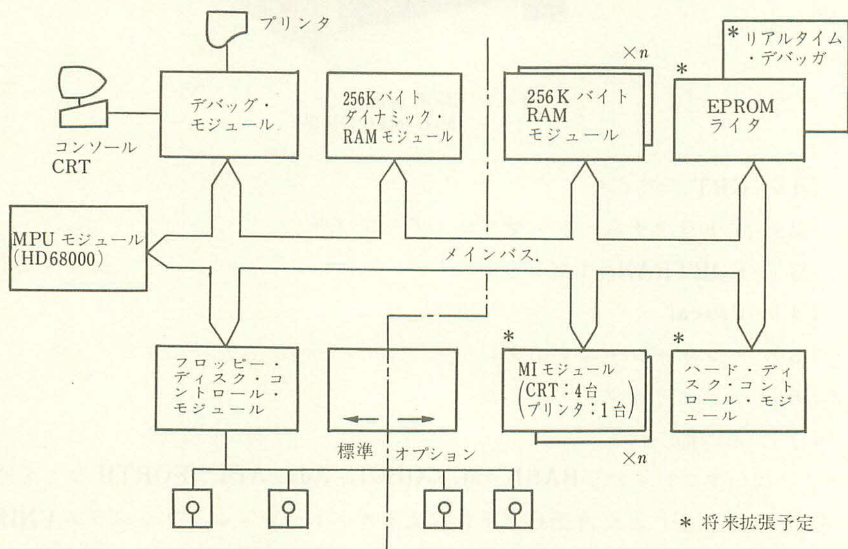


図 4.8 H680SD300内部モジュール構成（(株) 日立製作所の提供）

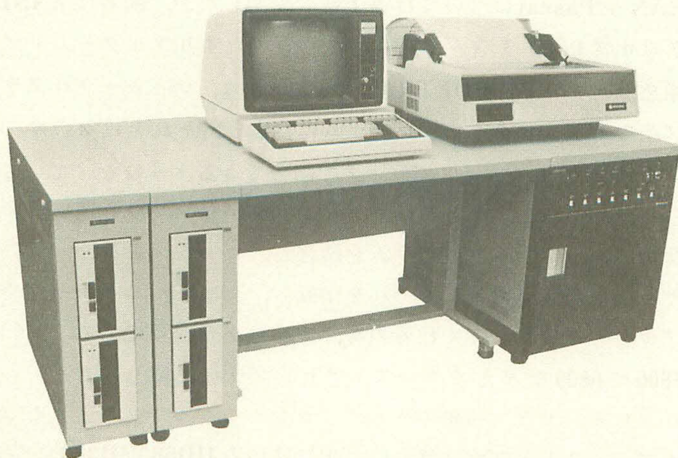


図 4.9 H680SD300

あるが，もう1組を追加して2Mバイトとすることができる．

プリンタは1行132文字のピンフィード・トラクタ方式で180字/秒，100行/分の速度をもつシリアル・プリンタである．1字は9×7ドットで構成され，アルファベットの大小文字とカナ文字その他を印刷でき，用紙幅は5インチから16インチで，セントロニクス・インタフェースで接続される．

コンソールは14インチのグリーンCRTで25行×80文字を表示できる．ただし25行目はエラー・メッセージやキャラクタ・インサート・モードの表示に使われているので，実質は24行×80字である．

右側の本体には15スロットのモジュールを収容できるが，標準はMPUモジュール，デバッグ・モジュール，フロッピー・ディスク・コントローラ，256Kバイトのダイナミック・メモリの4枚である．したがってメモリだけなら256×11＝2816Kバイトを補充して3Mバイトとできる．バスはMotorola社のVERSA busと同等である．

このシステムにMIモジュール（multiterminal interface module）を追加すると，CRT 4台とプリンタ1台が追加できるが，この場合はそれに応じたメモリを追加しなければならない．現在のところEPROMライタ，ハード・ディスク，リアルタイム・デバッグ装置，ASE（adaptive system evaluator）および高速ライン・プリンタ（700行/分）を追加することができる．

オペレーティング・システムはフロッピー・ディスクに格納されたFDOSで，タスクとジョブおよびファイルを管理し，各種コンパイラとそのユーティリティを利用することができる．コンパイラとしてはS-PL/H，FORTRAN，Pascalその他がある．このコンパイラで作ったオブジェクト・プログラムはEXORmacsでも使える．

リンケージ・エディタはコンパイラやアセンブラによってでき上がった再配置可能なオブジェクト・プログラムを結合し，ロード・モジュールなどを作り上げるもので，外部レベル参照関係の解決とモジュールの結合が基本になっている．

またEMS（executive monitor system）という物理空間用のモニタがあり，プログラムのロードやデバッグを行なうことができる．これはSD300用に開発されたシステムで，アドレス・ストップやブレーク・ポイントの設定，解除および表示，プログラムの実行とトレース，メモリやレジスタの内容の表示と変更，プリンタによるハード・コピーの出力，メモリ・マップの切替え，メモ

リのテスト，FDOSのブート・ストラップを含んでいる．このEMSによりメモリを二重に使うことができる．

このシステムにはCRTを1台追加できるが，EMSを使っているときは1台しか使えない．いわゆるマルチ・ステーション・システムを作るにはMIモジュールが必要である．

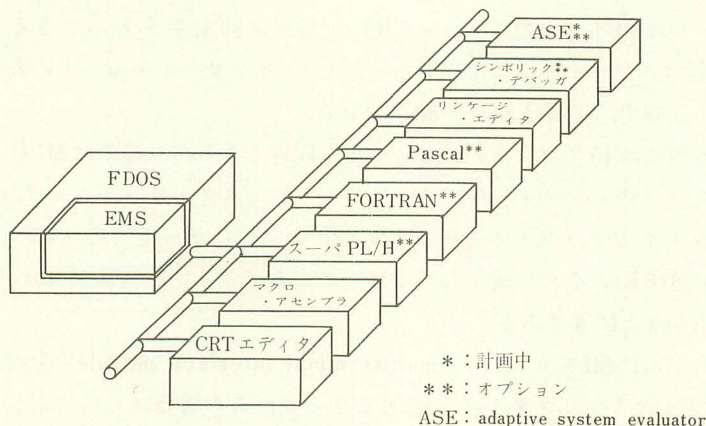


図 4.10　H680SD300 ソフトウェア構成（(株)日立製作所の提供）

4.2.3　μPDS D7800（安立電気（株））

これはマイクロプロセッサ開発システムμP development system D7800の略称で256Kバイトのメモリと12インチのCRTを標準装備しており，システム10，20，30の3種に分かれる．システム10はハードウェア・デバッグ用なので，ここでは主としてシステム20について解説する．システム30はシステム10，20の両機能をもっており，しかも4個のCPUを同時にエミュレートできる．システムの向上や拡張はもちろん容易である．参考までにブロック・ダイアグラムを付け加えた（図4.11）．

システム20は2台1組のミニフロッピー・ディスク（320Kバイト×2）を装備している（図4.12）．

キーボードは英数字カナその他が使えるが，そのほかに8個のソフトキーと11個の編集用キーをもっている（図4.13）．

増設用RAMとして128Kバイト，256Kバイト，512Kバイト，1Mバイトのものがあり，最大4枚まで実装できる．

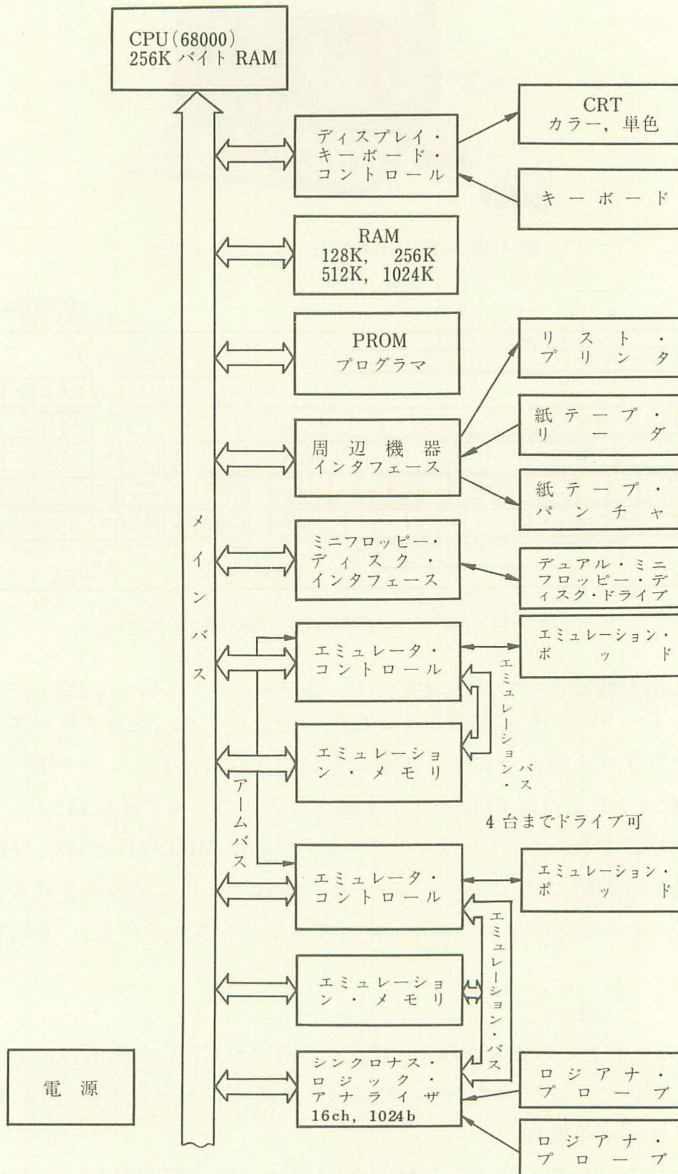


図 4.11 ブロック・ダイアグラム（安立電気(株)の提供）



図 4.12　μPDSD7800（安立電気(株)の提供）

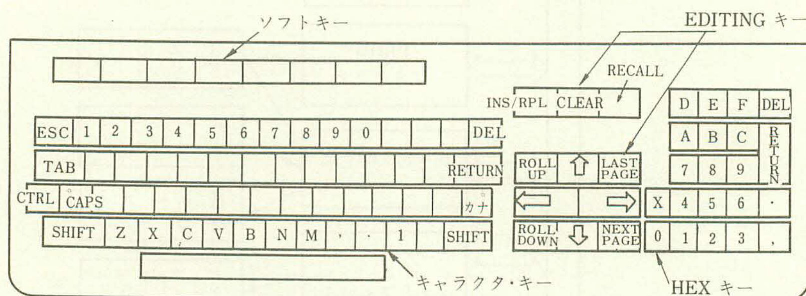


図 4.13　キーボードの構成（安立電気(株)の提供）

外部入出力機器としては，プリンタ，紙テープ・パンチャ，紙テープ・リーダ，カード・リーダなどがあり，これにエミュレータ，シンクロナス・ロジック・アナライザを組み合わせて使うことができる．エミュレータ用メモリとしては32Kバイトまたは64Kバイトを使うことができる（図4.14）．

ここに搭載されたオペレーティング・システムはMIOS/Uといい（図4.15），パワーオンと同時にフロッピー・ディスクからメモリに読み込まれるようになっている（そのためのイニシャル・ローダがセットされている）．その特徴をまとめよう．

（1）　ソフトキー

CRTのすぐ下にある8個のキーで，その内容がCRTの画面の最も下の行に表示されるので，コマンドを容易に選択して使うことができる（図4.16）．

（2）　ファイル管理

オペレーティング・システムMIOS/Uはすべての周辺装置，記憶装置をファイルとみなして管理する．したがってファイルはフロッピー・ディスクだけ

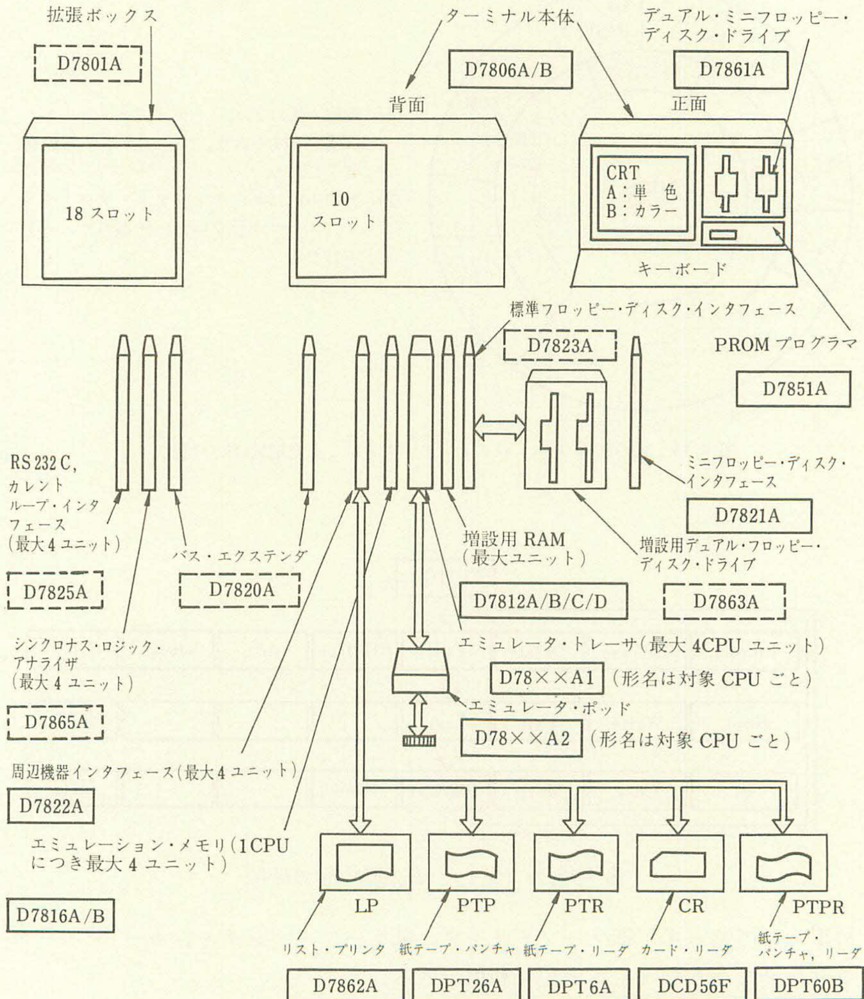
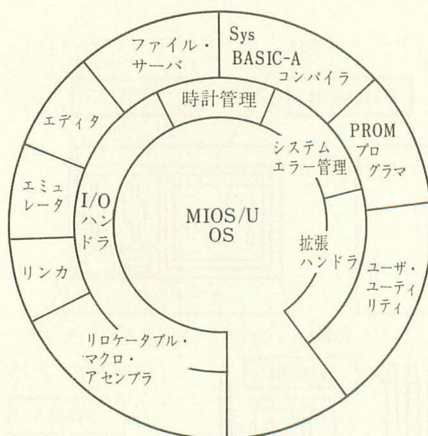


図 4.14 機器の組合せ例（安立電気(株)の提供）



(1) 内側の円はオペレーティング・システム（OS）の核であり，プログラムの実行管理を行なう

(2) 外側の円はユーティリティ・プログラムで，ユーザが行ないたい仕事のサポートを行なう

図 4.15　MIOS/U オペレーティングシステム（安立電気(株)の提供）

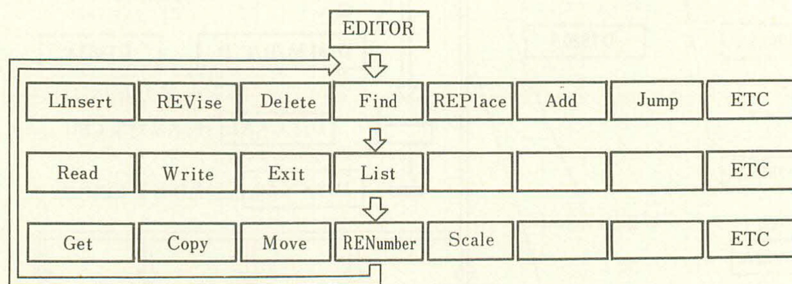


図 4.16　ソフトキー（安立電気(株)の提供）

に限られておらず，紙テープ・リーダ，紙テープ・パンチャ，カード・リーダ，プリンタなどもファイルとして利用できる．

（3） ユーティリティ

マクロアセンブラ，リンカ，エディタ，エミュレータ・サポート†，PROM プログラマ，Sys.BASIC-A コンパイラを備えている．マクロアセンブラやコンパイラによっていろいろのオブジェクト・プログラムが出力されるが，そ

† エミュレータはシステム 10，30 に限られている．

```
         ORG     30H
         JMP     RST65           ;JUMP RST65 PROCESSING AREA
         ORG     38H
         JMP     RST7            ;JUMP RST7 PROCESSING AREA
;
;****************************
;**  STATUS  INITIALIZE    **
;****************************
;
CLEAR:   LXI     H,GOIN          ;CLEAR WORKING AREA
CLEA1:   XRA     A
CLEA2:   MOV     M,A
         INX     H
         DCR     B
         JNZ     CLEA2
         STA     FTRAP           ;SET TRAP FLAG TO SYSTEM (0)
         MVI     A,0C9H
         STA     PORTO+1         ;INITIALIZE OUT INSTRUCTION AREA
         STA     PORTI+1         ;INITIALIZE  IN INSTRUCTION AREA
         MVI     A,0D3H
EDITOR
Read "FD1:DISPLAY.CRS"
Read  Write  EXit  List  ETC
```

図 4.17 ディスプレイ内のソフトキーの表示（安立電気(株)の提供）

の結合編集はリンカで行なわれる．このリンカは2パスで，エラー・メッセージをCRTに表示する．エラーは致命的なもの†と警告的なものに分かれ，前者は処理を中断して終了してしまうが，後者は処理を続行する．シンボル・リストとロード・モジュールのメモリ・マップを出力することができるので，プログラマには便利である．

Sys.BASIC-Aは，BASICを基調として制御用に開発されたもので，ファイル操作，ビット操作，文字列操作などが強化されており，わかりやすいという特徴をもっている．

このほかIntel社の8085用クロスアセンブラとZ80用クロスアセンブラがある．8086，6809などについては開発中である．

PROMプログラマは開発されたシステム・プログラムをROMに書き込むためのもので，エミュレーションが終了したプログラムを即時に書き込める能力がある．

† fatal error：決定的エラー．これが起こるとプログラムの結合を行なっても無意味になってしまう．

なお，時間管理ができるように内部クロック・シンセサイザをもっており，ソフトウェア実行時間の測定を正確に行なうことができる．

4.3 パーソナルコンピュータ，ビジネスコンピュータ，その他

いわゆるパソコンの定義は明確でない．パーソナルコンピュータと称して売られているものはほとんどが8ビットのマイクロプロセッサを使っており，カセットやフロッピーディスクを補助メモリとし，BASICを用いている．しかし最近はFORTHやPascal, FORTRANなどの言語も使えるようになり，科学実験処理その他を含めて，用途も大幅に広がった．とりあえずここでは，パーソナルコンピュータと名付けられているものを含め，筆者の主観でまとめることとした．計測用として使えるはずのものさえも含まれてしまっていることに「何か」を感ずる人も多いと思うが，御了承を願いたい．

68000を用いたパーソナルコンピュータとしては安立電気（株）のPACKET II，伊藤忠（株）の680/20ビジネスコンピュータPRO-Ⅳ，JUN社のステーション4D，Tandy Radio Shack社のTRS-80モデル16，Xerox社のStar，Apollo社のDomain，Alpha micro社のAM-106Z，Corvus system社のCorvus Concept，Cromemco社のCromemco One，Motorola社のVMC 6812，横河・ヒューレット・パッカード社の9826A，その他多数のシステムがある．その全部を解説することはもちろんできないので，なるべく国内で作られているシステムに限り，これに筆者の独断を加え，また，関係各社をまわって集めた資料を基にして書き上げた．

オペレーティング・システムを中心として分類すれば，次の四つになろう．Motorola社および（株）日立製作所の開発したもの，CP/M，UNIX，その他独自に開発したもの．

CP/Mによるシステムはまだ非常に少なく，UNIXによるものは多い（少なくとも1982年10月の時点では1983年からはCP/Mによるシステムも増加しよう）．Motorola社および（株）日立製作所のディスク・オペレーティング・システムについてはすでに紹介したので，ここでは取り上げない．

4.3.1 PACKET II（安立電気（株））

68000 L8を主プロセッサ，Z80を周辺コントロール用プロセッサ用とするシス

テムで，16Kバイトの ROM と 256Kバイトの RAM を標準装備する．9.5インチのグリーン CRT を使い，21行×64文字を表示できる．文字としては英数字，カナ文字，ギリシア文字を含み，オプションでグラフィックス機能を付加することができる．

補助メモリとして約300Kバイトの容量をもつミニフロッピー・ディスク1台を標準装備しており，もう1台を内蔵できる．このほか，35文字/行の感熱式プリンタを装備しており，CRT 画面のハードコピーはもちろんキー一発でできる．その例を示す．

READY
FFT パワースペクトラム
LEVEL
0
−10
−20
0　100　200　300
FREQUENCY

(a)

```
PPPP                           k
P   P                          k   k              t
P   P     aaa       ccc        k   k     eee     ttt
PPPP         a     c           k k      e    e     t
P         aaaa     c           kk       eeee       t
P        a    a    c     c     k k      e          t
P         aaaa       ccc       k   k     eee       tt t
```

```
   66       888       000       000       000
  6        8   8     0   0     0   0     0   0
 6         8   8     0   0     0   0     0   0
 6666       888      0   0     0   0     0   0
 6    6    8   8     0   0     0   0     0   0
 6    6    8   8     0   0     0   0     0   0
  6666      888       000       000       000
```

(b)

図 4.18　画面表示とそのプリントの例（安立電気(株)の提供）

背面に7個のスロットがあるので，メモリや周辺機器の増設に使える（もしメモリだけに使うとすれば128Kバイト×6=768Kバイトの増設が可能である）．下にブロック・ダイアグラムとPacketIIの写真を示す．

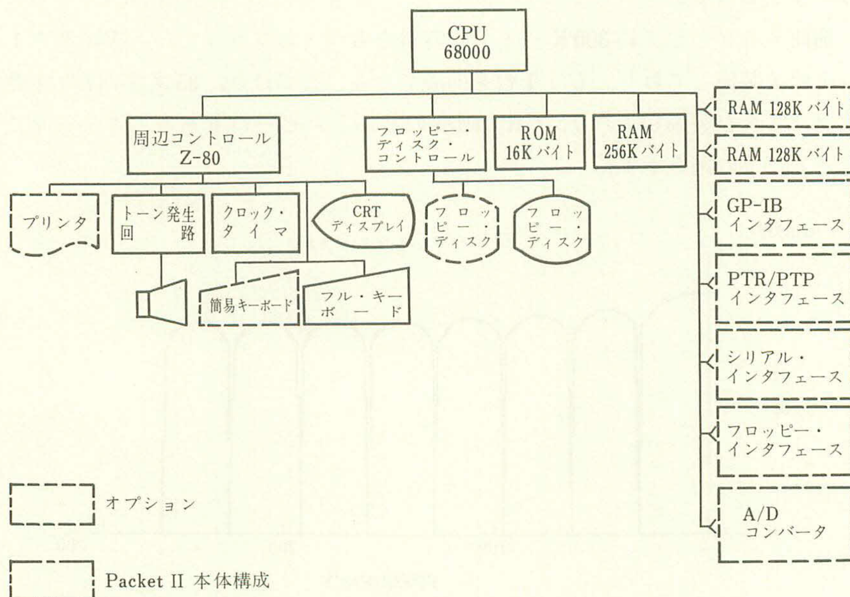


図 4.19　PacketIIのブロック・ダイアグラム（安立電気（株）の提供）

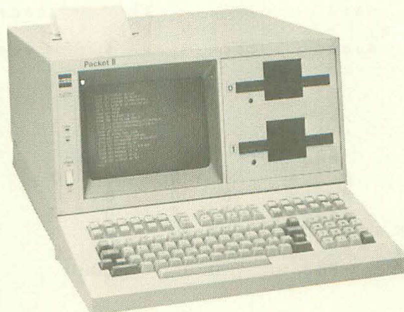


図 4.20　PacketII（安立電気（株）の提供）

キーボードはいろいろなキーを含み，機能も豊富である．フロッピー・ディスクを用いたオペレーティング・システムFDOSを搭載し，BASICを主要言語とする．このほかUCSD p-systemを用いることもでき，ANSI-77に準拠したPascalとFORTRANを使うこともできる．COBOLやAdaもいずれ使えるようになろう．

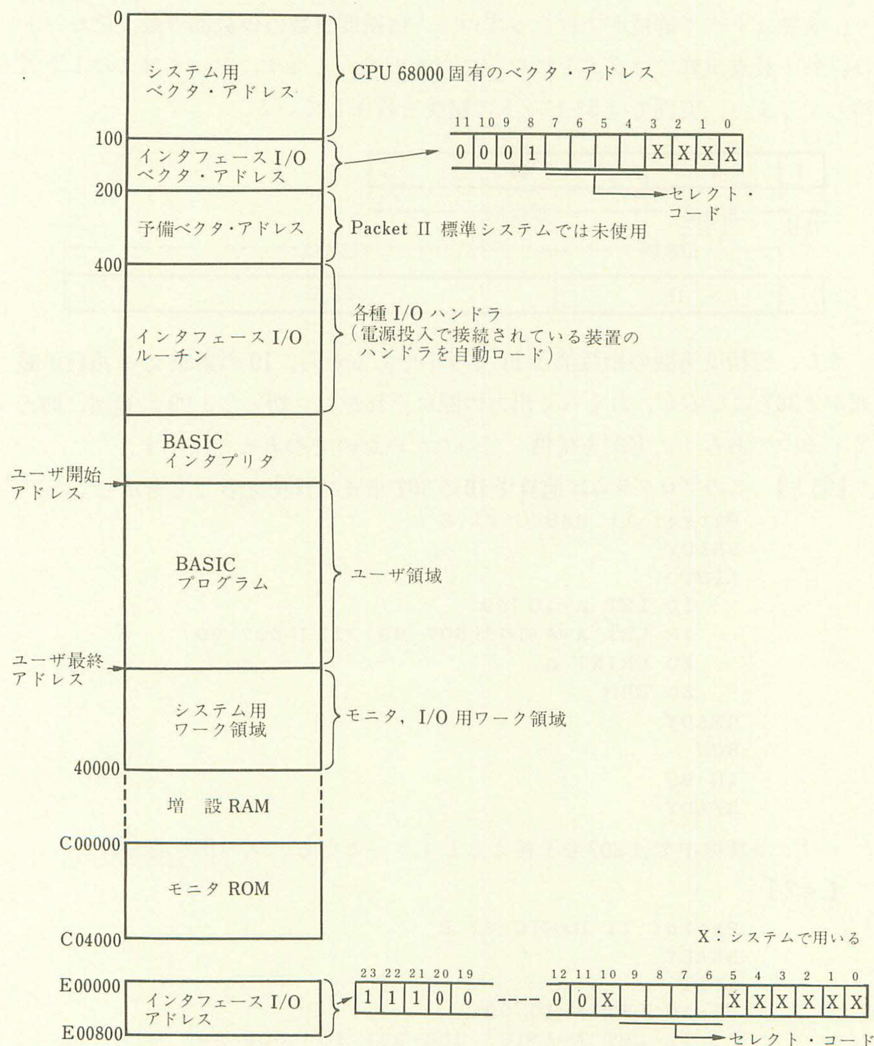
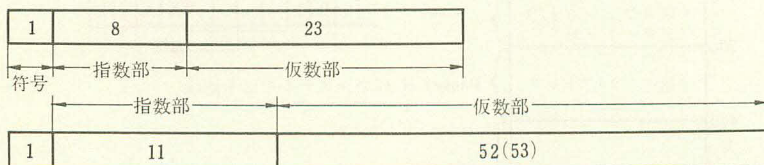


図 4.21 メモリマップ（安立電気(株)の提供）

BASICインタプリンタは安立電気(株)で開発したもので，約100Kバイトの大きさを有し，残りの約150Kバイトをプログラム・エリアに当てることができる．メモリ・マップは前ページの通りである．

このBASICでは2バイトの整数(−32768〜32767)，4バイトの単精度実数(6桁の仮数$\times 10^{\pm 33}$)，8バイトの倍精度実数(12桁の仮数$\times 10^{\pm 99}$)を用いており，演算はすべて倍精度で行なっている．倍精度実数の仮数部の最上位ビットは浮動小数点演算では必ず1になっているので，メモリにおいてはこの1を省略してしまい，実質では53ビットの精度を確保している．



また，倍精度実数の指数部は11ビットであるから，10の累乗では指数の範囲が±307になるが，おそらく出力の際にそれを打ち切って±99の範囲に抑えているのであろう．実例を提供していただいたので参考までに示す．

【例1】 このプログラムは演算中10の307乗までは使えることを示している．

```
Packet II BASIC V1.8
READY
LIST
    10 LET A=10↑99
    12 LET A=A*10↑(307-99)/10↑(307-99)
    20 PRINT A
    30 END
READY
RUN
1E+99
READY
```

一方，演算途中で±307乗を越えるとエラーとなる．次の例を見られたい．

【例2】

```
Packet II BASIC V1.8
READY
LIST
    10 LET A=10↑99
    12 LET A=A*10↑(308-99)/10↑(308-99)
    20 PRINT A
```

```
    30 END
READY
RUN
ERROR 1002 AT LINE 12
```

これは行12で10↑308を計算するためオーバフローを生じたことを示している．

最大の特徴は，実行時間の空きを利用した並列処理にある．実際，Packet IIは最大四つのプログラムを並列処理できるので，時間を大変有効に利用しているといってよい．もちろん実行の優先度も有効で，パーソナルコンピュータとしては素晴らしいシステムといえよう．ライブラリの一部と実行時間の測定例を示す．

表 4.2 実行時間

加 算 ＋	1.14 m秒	累 乗 ↑	18.2 m秒
減 算 －	1.12	正 弦 SIN	9.6
乗 算 ＊	1.48	余 弦 COS	9.6
除 算 ／	1.64	対 数 LOG	9.4

ライブラリの一部：

（1） 統計，I，IIの二部に分かれ，Iは平均，偏差，正規分布検定，t分布検定など，IIは分散分析などを含む．

（2） 複素数，行列などはもちろん，代数方程式や数値積分，常微分方程式についてのサブルーチンもある．フーリエ解析FFTについてもBASICおよびアセンブラによるサブルーチンが用意されており，その演算時間は1024点でBASICによるものが100秒，アセンブラによるものが5秒を要するという．最後にベンチマーク・テストに用いたプログラムの例を示す（実行時間は3回の平均値である）．

```
10 PRINT "START"
20 FOR K=1 TO 10000
30 NEXT K
40 PRINT "END"
50 END                  (1.2秒)
```

```
10 PRINT "START"
20 FOR I=1 TO 1000
30 LET A=RND(1)
40 NEXT I
50 PRINT "END"
60 END                  (2.6秒)
```

```
10 PRINT "START"
20 FOR I=1 TO 1000
30 LET A=LOG10(123)
40 LET A=LOG(456)
50 LET A=EXP(7)
60 NEXT I
70 PRINT "END"
80 END                    (23.2秒)
```

```
10 PRINT "START"
20 FOR I=1 TO 100
30 FOR J=1 TO 10
40 LET A=SIN(J)
50 LET A=COS(J)
60 LET A=TAN(J)
70 NEXT J
80 NEXT I
90 PRINT "END"
100 END                   (34.4秒)
```

4.3.2 Personal-Graphics-Station 4D (JUN社)

これは大変面白いシステムである．JUN社の堀北司氏がボードのラッピングから始めて独力で作り上げたシステムで，別に相談したわけではなかったがMotorola社のVERSA moduleと似たモジュールを使い，ピン配置はそれに合わせてある．

JUN社は1918年に創業されたファッション関係の会社であるが，ライフスタイルが時代とともに変わるのをファッションという角度からとらえようとして努力し，いろいろな事業に進出してきた．現在ではワイン，音楽，ゴルフ，インテリアなどに関係している．

4Dもその一つで，クリエーティブ部門にエレクトロニクスを導入して現代化をはかり，作業の効率化や省力化を進め，"デジタル感覚"といわれる新世代感覚をファッションとしてとらえようとしている．この世界でははなはだ進歩的な考え方であろう．たとえば洋服のモード，スタイルは時代とともにどんどん変わる．洋服のスタイル，色彩などをさまざまに変えたとき，そのサンプル数は膨大なものになり，手間も大変である．これをブラウン管(CRT)上で簡単に再現し，プリントでき，しかもこのデザインを3次元的につかまえられるとしたら，単なる省力化だけではない何かをファッションの世界に導入できる

ことになろう．筆者はファッションについては全くの素人ではあるが，こういうことは何となくわかるような気がする．

さて4Dを解説しよう．上のような狙いから，当然コンピュータ・グラフィックスが登場してくる．そこで14インチの高解像度カラー・モニタを基本構成に含めた．もちろん主プロセッサは68000で，8インチのフロッピー・ディスク（1Mバイト，IBMフォーマット）を内蔵し，64Kバイトのシステム・メモリをもっている．

最大の特徴は1600万色にのぼる豊富な色彩で，画像や色彩の組合せは自由に変えられる．このグラフィックスのため512Kバイトの画像メモリを別に設け，専用のLSIで制御し，512×512に達する画素で表示するようにした．実際に必要な色は200もあれば十分なので，現在は1600万色の中から256色を任意に選択するようになっている．

描画速度は1画素当たり0.8μ秒～1.5μ秒である．画像の変形や色の変化は簡単にできるので，ファッションにはもちろん使える．この画像をカラーでプリントし，VTRや35ミリフィルムに収められるようになっているので，テレビのコマーシャル・フィルムの制作には大変有効である．このため4D本体とのインタフェースとしてシリアル2本，パラレル4本，アナログ16本が内蔵されている．

オプションとして，本体のシステム・メモリを256Kバイト単位で1Mバイトまで拡張し，プリンタを設置し，20Mバイトのハード・ディスクを増設することができる．

グラフィックスの入出力面には細心の注意が払われており，20インチ高解像度カラー・モニタを含めて，現在9種類のオプションがある．将来の拡張が大変楽しみなシステムといってよい．ブロック・ダイアグラムを次に示す．

ソフトウェアも大変ユニークである．FORTH 79にグラフィックス機能を強化したFORTH系言語ØDLに基づき，システムが構成されている†．図形データのファイリングなどのため，データ・ベースØDBがあり，ØDLで記述されている．なおØDLはクロス・コンパイルによって68000のメモリに導入された††．

† ØDL＝ØD language，DB＝data base．これは筆者の想像であるが，ØD，1D，2D，3DのDはdata，digital，developementなどのDであろうか．なおFORTHはFORTH社の登録商標である．

†† 現在は，68000上のØDLで言語そのものが完全に記述されており，他機種へのターゲット・コンパイルも可能である．

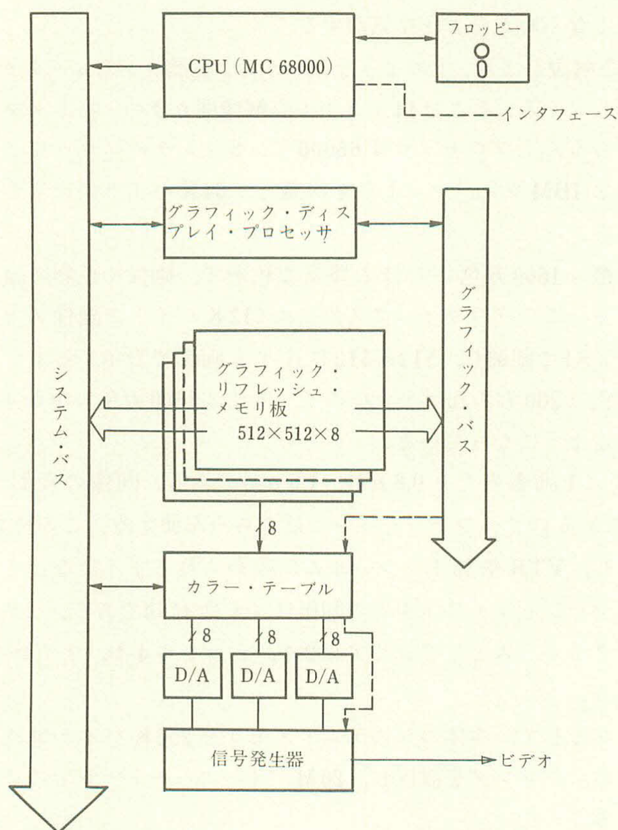


図 4.22　ブロック・ダイアグラム

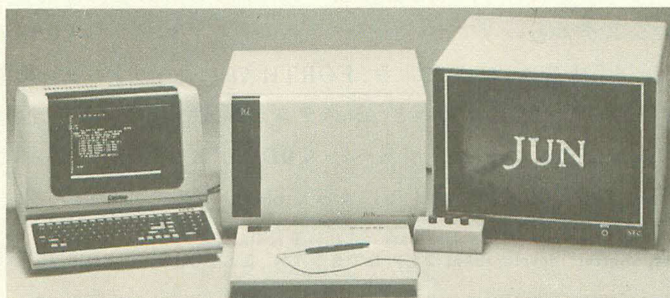


図 4.23　4Dの基本システム

平面図形や立体図形のためのグラフィックス・パッケージがそれぞれ2D，3Dである†．図形の回転，スケーリング，平行移動，座標変換，立体図形の透視などが含まれており，ØDLで記述されている．

ØDLはいろいろな特徴をもっているが，その一つとして会話型で構造化プログラミングの精神に満ちていることがあげられよう．自己拡張機能もあり，マトリクス演算も行なえ，データ型の定義も自由自在なので，多方面に応用できる．

実際にシステム4Dはインテリア，服飾，建築，テキスタイルなどのデザイン分野で使われている．すなわち形や色をその場で自由に変化させてそのベストを決められるので，ダミーやサンプルの製作に要した時間や手間は大幅に節約できる．

また，アニメーションにも使え，従来のようにセルシートに1枚ずつ書いて作っていた動画作成が大幅に合理化できる．すなわちセルシートが1枚書いてあれば，それをカラー・モニタに映し出し，その部分部分を自由に動かすことができる．したがってアニメーションのみならず，天気図の作成，ビジネス用シミュレーションの表示，幼児教育への応用などへの進出が期待できよう．音楽分野ではシンセサイザと連動したステージ効果の予測なども可能で，その応用面は非常に広い．

4.3.3 パーソナル・テクニカル・コンピュータHPシリーズ200モデル16/26/36（横河・ヒューレット・パッカード社）

68000を用いた科学技術用システムとして，YHP（横河・ヒューレット・パッカード）社はHPシリーズ200モデル26を1981年から，モデル16/36を1982年から発売している（図4.24）．その要点を述べよう．

モデル26はCAT (computer aided testing) を主目的としたシステムで，7インチのCRT，260Kバイトのミニフレキシブル・ディスク1台，64Kバイトのメモリを標準装備している．

メモリは，約2Mバイト（別売のバス・エクスパンダを使えば約7Mバイト）まで実装できる．このうち，ユーザ・エリアは使用言語で変化する．たとえば2Mバイトを実装して，BASIC 2.0を使った場合は約1.8Mバイト，Pascal 2.0を使った場合は約1.9Mバイトがユーザ・エリアとなる（RAMベースの場

† 2DはTwo dimensionという意味もあろうかと想像する．

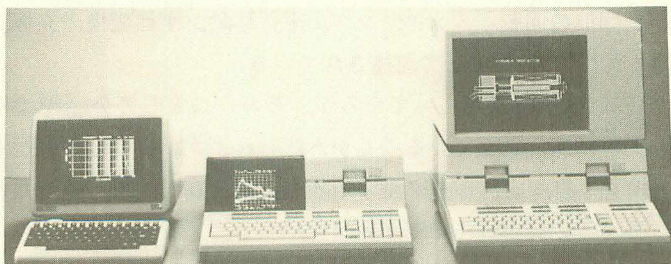


図 4.24　YHP 200 モデル 16/26/36

合).

ミニフレキシブル・ディスクは260Kバイトが使える．CRTディスプレイは25行×50文字を表示でき，グラフィックの分解能は，400ドット(横)×300ドット(縦)である．ディスプレイ用バッファとして文字用に2Kバイト，グラフィック用に16Kバイトをもつ．文字コードは8bit JISに準拠しているが，ユーティリティ・ソフトウェア((株)ユニック製)により，漢字も表示できる．このほか，HP-IB (Hewlett-Packard interface bus)を内蔵しており，これによって130種を越える各種機器を接続することができる．HP-IBはIEEE標準488-1978の基本になった最も汎用性の高い方式である．

モデル36はCAE (computer aided engineering)を主目的としたシステムで，12インチのCRT，ミニフレキシブル・ディスク2台，メモリ64Kバイトを標準装備している．CRTは，25行×80文字を表示でき，グラフィックの分解能は512ドット(横)×390ドット(縦)である．ディスプレイ用バッファとして文字用に3Kバイト，グラフィック用に24Kバイトのメモリをもつ．上記以外の点は，モデル26，36ともにハードウェア上もソフトウェア上も全く共通な仕様となっている．したがってメモリ，インタフェースなども完全な互換性を保っている．

さらに，モデル26/36には，停電保護ハードウェアが取付け可能で，1分間までの停電，電源ラインの瞬断に対する保護が行なえる．

モデル16は上記のモデル26/36の特徴を生かしたまま技術者が個々に使用できるように，従来のデスクトップという思想をさらに推し進め，コンパクトと廉価をさらに追求したモデルで，個々の技術者単位でのCAT，CAEのアプリケ

ーション，データ整理，統計処理，スケジュール管理などを主目的としたシステムである．9インチCRT，128Kバイトまたは512Kバイトのメモリを標準実装している．マス・ストレージは，後述のSRMのノードとして使用することも考慮して別売となっている．キーボードは分離型とし，小さな机の上でも使用できるよう工夫されている．

メモリは約0.7Mバイト（別売のバス・エクスパンダを使えば約4Mバイト）まで実装できるが，使用言語によるユーザ・エリアの変化は他のモデルと同じである．CRTディスプレイは25行×80文字を表示でき，グラフィックの分解能は400ドット（横）×300ドット（縦）である．ディスプレイ用バッファとして文字用に3Kバイト，グラフィック用に16Kバイトをもつ．ほかの仕様は，停電保護ハードウェアがないこと，メモリ/インタフェース・カード用スロットがモデル26/36が8であるのに対し2になった代わりに，RS-232-C（19200/bps）がHP-IBインタフェースとともに内蔵されたこと，キーボードの小型化に伴って，使用頻度の少ない一部のキーが省略されたことを省けば，モデル26/36とハードウェア的にもソフトウェア的にも共通である．

表 4.3 言語の要点

	BASIC 2.0	HPL 2.0	Pascal 2.0
実数	整数（16ビット） −32768〜32767 実数（64ビット，指数部11ビット） だいたい$10^{\pm 308}$の範囲をカバーする	実数（64ビット，指数部12ビット） だいたい$10^{\pm 511}$の範囲をカバーするが，表現は$10^{\pm 99}$の範囲で行なう	整数（32ビット） −21,4748,3648〜 21,4748,3647 実数（64ビット，指数部11ビット） だいたい$10^{\pm 308}$の範囲をカバーする
一般関数	絶対値，三角関数，平方根，自然対数など 種類，数は言語により異なる BASIC 2.0，HPL 2.0では30種以上		
文字列関数	数値 ⟷ 文字，文字列の長さなど7種以上 記号，数，種類などは言語により異なる		
ステートメント	ステートメントは大変豊富である 入出力機能も完備している		
その他	BASIC拡張機能2.0を使用すると，190種以上のステートメント，コマンド群が追加される	デスクトップ・コンピュータ9825のHPLと上位互換性がある	UCSD Pascalのほとんどを包含する

さらにシリーズ200のすべてのモデルには，ロータリ・ノブがついている．これは，カーソル制御，グラフィック入力に使えるだけではなく，コンピュータへのアナログ的データの入力も可能にしている．

ソフトウェアはモデル16/26/36に共通である．言語としては，BASIC 2.0，HPL 2.0，Pascal 2.0があり，1台のコンピュータの上で，アプリケーションに合わせて切り替えて使用することができる．特に，ROMベースのHPL 2.0，BASIC 2.0の言語システムの場合には，それぞれのROMボードをコンピュータに実装しておき，電源をONにしたときに，どちらのシステムを使用するかを選択できるようになっている．

これらの言語システムの特徴を表4.3，4.4にまとめた．

また，1982年11月より発表されたBASIC拡張機能2.0を使用すると，メモリのユーザ・エリアは多少減るが，BASIC 2.0の機能は大幅に拡張される．その中には，マトリクス演算，プログラム編集を容易にするコマンド群，入出力操作を高速に行なうためのバッファ命令群が含まれている．

さらにアプリケーション・ソフトウェアも豊富に用意されており，統計処理，波形解析といった科学技術計算用のユーティリティや，グラフィック・プレゼンテーション，Visi-Calcといったデータ整理，発表用のソフトウェアもある．

インタフェースは，HP-IBも含め7種類が発売されており，種々の機器との接続が可能であるように考慮されている．また，その制御は容易で，ほとんどのインタフェースでは，同一の入出力命令が使用できるよう工夫されている．

グラフィック処理機能も強力で，プログラムの変更なしで，CRTへもプロッタへもグラフィック出力が行なえるユニファイド・グラフィック機能がある．

表 4.4　実行速度

	BASIC	HPL		BASIC	HPL
絶対値	0.041 m秒	0.120 m秒	余　弦	4.200 m秒	16.25　m秒
整数化	0.090	0.40	加　算	0.137	0.226
平方根	1.743	4.433	減　算	0.161	0.309
乱　数	0.419	0.920	乗　算	0.302	0.400
正　弦	3.621	16.39	除　算	0.432	0.420

データはすべて実数における演算の代表値である
Pascal 2.0は，BASIC 2.0やHPL 2.0によるプログラムと比べて3～20倍速い

なお，横河・ヒューレット・パッカード社では，HP シリーズ200 コンピュータおよびデスクトップ・コンピュータ9845で使用できるネットワークとして，SRM (shared resource management) を1982年より発表している．これは最大63台までのコンピュータでハード・ディスク，プリンタなどを共有する形式のローカル・エリア・ネットワークである．

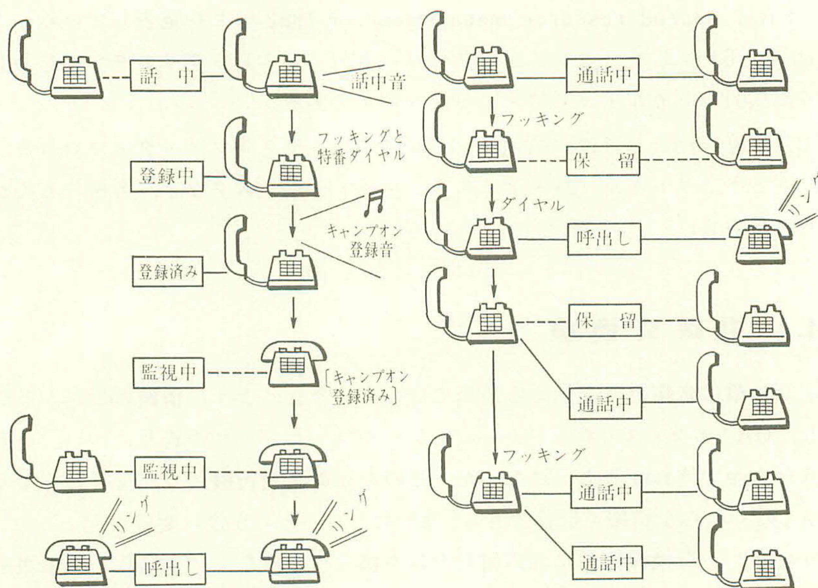
周辺機器の例としては，高速シリアルプリンタやグラフィック・プロッタ，タブレット，フレキシブル・ディスク，ハード・ディスクなど，各種のものがそろっている．

4.4 電話交換機

電話は単に音声だけを伝える道具ではない．今日のように情報の伝送が多様化し，OA（オフィス・オートメーション）やファクシミリの普及に伴い，いろいろな方面で使われるようになった．そのため電話交換機にも“より安くより小さくいろいろな情報を伝達できる”ようにしたいという強い要望が起こった．こうなると，交換機は単に音声信号だけを扱うのではなく，多様化した情報を

表 4.5 DX 規格の概要

		DX20	DX20L	DX30	DX40	DX50
内線容量 (回線数)		128	256	512	1024	5120
ネットワーク	チャネル数	32		32	32	
	ハイウェイ数	4		8	16	
冗長化		一重		一重/二重		二重 (分散制御)
キャビネット	幅	650 mm	1580 mm	740 mm		680 mm
	奥行	500	500	600		680
	高さ	1300	1950	1950		2150
ネットワーク構成		タイム・スペース・タイムスイッチ T-S-T				タイム・スペース・スペース・タイムスイッチ T-S-S-T
その他		コーデック：シングル・チャネル，自然空冷				



（a） 内線相互キャンプオン・サービス例
内線相互発信で相手が話中の場合，フッキングを行なうとキャンプオン登録音が聞こえる．登録後，自内線と相手内線の両方が空きになると自内線を呼び出し，応答すると相手を呼び出す．登録は登録後一定時間（約5分）が経過した場合解除される

（b） 内線アッドオン・サービスの例
内線と通話中にダイヤル操作でもう一つの内線を呼び出して，3内線による通話が行なえる．内線相互接続だけに適用する

図 4.25　サービスの例

速く確実に高品質化して送らねばならない．

現在では音声とデータ信号をともに扱えるPCM (pulse code modulation) 方式のディジタル交換機が主流になっている（PCMというのは信号を細分して，ビットで表現できるようにしてしまい，ディジタル信号として通信する方式をいう．これによって音質の向上，確実な受信が可能になるので，信頼度などが大幅に改善される）．こういう背景のもとに，（株）日立製作所は主プロセッサとしてHD 68000を使ったディジタル電子交換機DXシリーズを開発した．

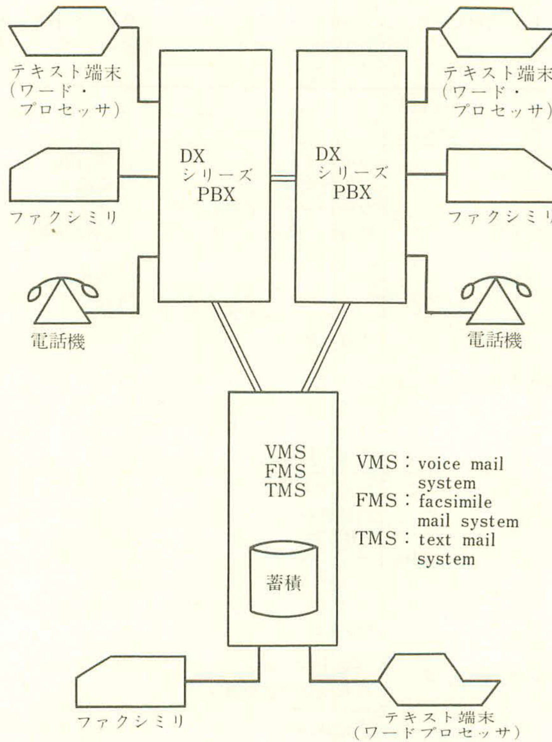


図 4.26 各種メール・システムとの構成例　ボイス・メール，ファクシミリ・メール，テキスト・メールなどのOA用メール・システムと結合し，広範囲なネットワーク・システムが構成できる

DXシリーズの概要を述べよう．これにはDX20，DX20L，DX30，DX40，DX50という5機種がある．いずれもアナログ信号をディジタル信号に変換し，交換動作をディジタルで行なうもので，プロセッサに68000を用い，フロッピー・ディスクをバックアップ・メモリとしている．符号化形式はCCITT標準のμ-law方式で，国内のPCM伝送方式に合わせている†．扱いうる回線数も128本から5120本までと各様である（表4.5）．

このDXシステムのサービス機能を表4.6に示した．

† CCITTは国際電信電話諮問委員会 international telegraph and telephone consultative committy，μ-lawは企業規格を意味する．

表 4.6 PBXシステム「DXシリーズ」のサービス機能（内線電話機や局線，中継台に対し操作性を向上させるとともに，各種応用サービスの充実が図られている

項目		内線	局線，中継台
操作性向上	話中対策	リセット・コール 内線代表 コール・ウェイティング 内線相互キャンプオン	半自動キャンプオン 自動キャンプオン アイドルフラッシュ 話中内線割込み オートマチック・リコール 内線話中ランプ盤
	不在対策	コール・ピックアップ 不在転送 ページング ページング・トランスファ ポケットベル ポケットベル・トランスファ 不応答転送	——
	ダイヤル操作	短縮ダイヤル 内線群短縮ダイヤル PB内線 ホットライン いっせい指令 ラストナンバリダイヤル OGキューイング	トランク指定接続 ワンタッチ・コール ワンタッチ・ダイヤル
多者通話化		内線アッドオン 内線制御多者会議	——
着信時の利便化		着信音識別 イミディエート・リング 内線着信規制	完全着信順応答 台間連絡転送
通信経費低減		サービスクラス1回線単位 特定地域市外制御 3分予報音 専用線発着信接続 専用線タンデム接続	——
その他		緊急呼出し電話 コールホールド	保留音送出 シリーズ・コール 夜間自動切替え 夜間受付台 夜間トーキーサービス 自動共電内線収容 中継台ディジタル表示 オートレリーズ 即時/待時依頼発信

PB：プッシュボタン，OG：アウトゴーイング，C/I，C/O：チェック/イン，チェック/アウト，CPU：コンピュータ

保守運用	ホテルなど	その他
――	――	多機能電話機
――	――	
――	ハウスホン	
――	――	――
――	客室着信規制 客室間接続規制	――
通話明細出力 内線別度数登算 グループ別度数登算	客室課金	――
ラインロックアウト ナンバグループ テナント トーキーサービス 出力路相互間う回接続 内線延長 幹部秘書 トラヒック測定 各種障害データ出力 各種接続試験ほか	モーニングコール ルーム・インジケータ サービス回線 客室番号3～5桁混在 C/I, C/O時サービスクラス変更 ルームチェンジ ホストCPUとの接続 メッセージ・サービス	データ・プライバシー ボイス・メール ファクシミリ・メール テキスト・メール ディジタル・インタフェース

サービス機能として電話機利用者の利便性向上手段，通信経費の低減手段，ディジタル系などの各種サービス機能との結合を考えており，図4.25に示すような例がある．

表4.7に示すように障害種別に応じたシステム再開機能もある．将来の交換機はもっと多くの情報を扱えるようになろう．

表 4.7 PBXシステム「DXシリーズ」のシステム再開機能　（障害種別に応じた段階的システム再開機能を備えているため，交換サービスの連続性が確保できる）

システム再開名称	再開方法	特徴
フェーズ0	障害検出時，リトライまたは障害装置をシステムから切り離して障害検出点から再開する	交換動作への影響全くなし
フェーズ1	システムに重要なエラーが発生し，正常な運転ができない場合，予備系のプロセッサに切り替わる	通話中の呼出しは，通話接続できる
フェーズ2（自動）	フェーズ1が一定時間内に多発した場合や切り替わるべき予備系プロセッサが準備未完了の場合，システム全体を初期設定し再開する	システム全体を初期設定する（加入者クラスなどはほぼ最新の状態が引き継がれる）
フェーズ2（手動）	交換機の運転開始時などに，保守者が手動でフェーズ2システム再開をする	同　上

このソフトウェアを開発するに当たっては，Hitac Mシリーズ・コンピュータ・システムをクロス・ホスト・マシンとして用いることにし，システムの基本構成を図4.27のように考えた．実用化の基本方針は次の通りである．

（1） 通信サービスおよびプログラム仕様の言語化と図式表現の実用化——CASTLE†とPDL/PAD††の開発

（2） リアルタイム・システムむき高級言語の開発——S-PL/H，プログラミング支援ユーティリティUTL 68†††の開発

（3） シミュレータによるプログラム・モジュール・テストの標準化，テスト項目作成の効率化——HITS†4とAGENT†5の開発

（4） ソフトウェア運用管理作業の効率化——SGN†6，DGN†7，SLIM†8の開発

† call state transition language.
†† program description language/problem analysis diagram.
††† utility for HMCS68000.
†4 highly interactive testing and debugging system.
†5 automated generator of externel test-cases.
†6 system generator.
†7 office and subscriber data generator.
†8 system library management system.

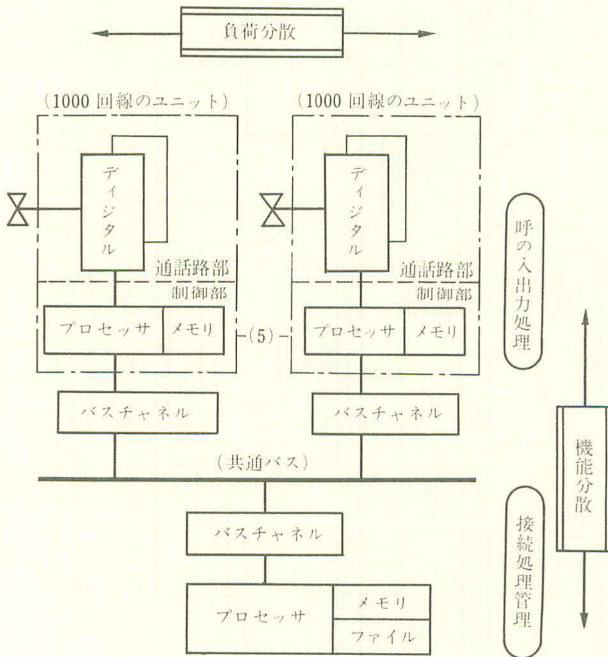


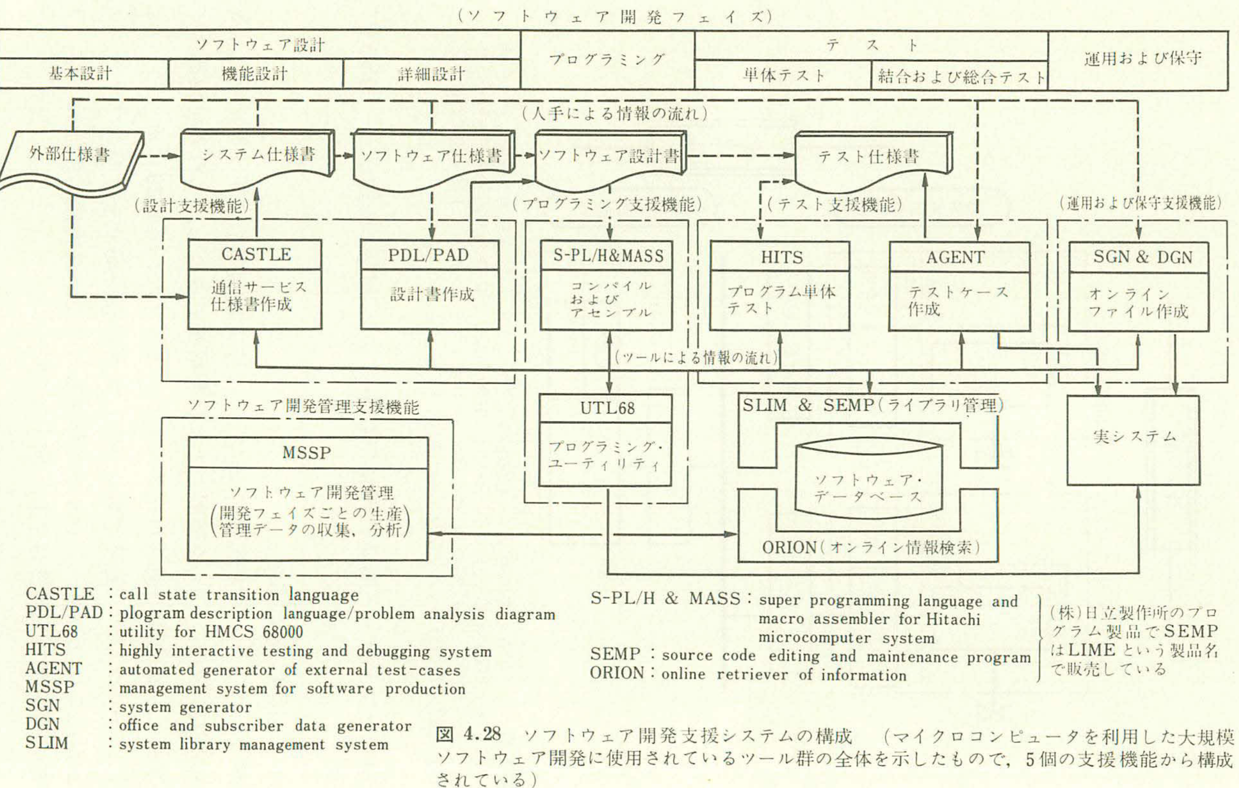
図 4.27 システムの基本構成 1000 回線以下のシステムは，1 個のプロセッサで制御を行なうが，1000 回線以上のシステムでは，1000 回線ごとに呼の入出力処理を行なうプロセッサを配置し，接続処理，管理を行なうプロセッサをシステムで 1 個共通に設置する負荷，機能分散制御形マルチプロセッサ方式としている.

(5) ソフトウェア開発管理情報収集および分析の迅速化 —— MSSP† の開発

これらを図 4.28 に示す.

実際にソフトウェアを開発するに当たっては交換機をブラックボックスと考え，外部インタフェースを種別ごとに仮想機械とみなし，これらを仮想スイッチで接続し，端末間の通話がなされるものと考えた．図 4.29 にその例を示す．さらにソフトウェアの実現に当たり実行頻度の高い部分をアセンブラで，プログラム量の多い部分をコンパイラで作成したので，交換機の性能およびソフトウェアの信頼性と生産性は著しく向上している.

† management system for software production.

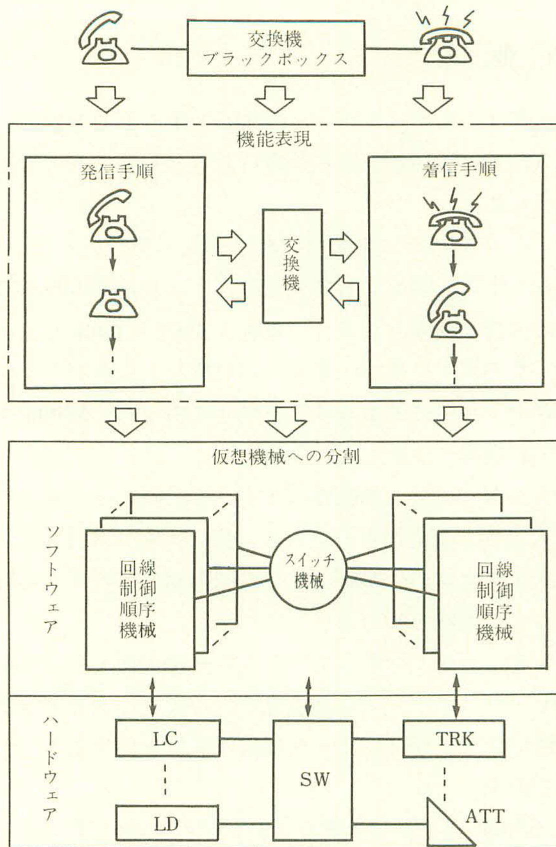


CASTLE : call state transition language
PDL/PAD : plogram description language/problem analysis diagram
UTL68 : utility for HMCS 68000
HITS : highly interactive testing and debugging system
AGENT : automated generator of external test-cases
MSSP : management system for software production
SGN : system generator
DGN : office and subscriber data generator
SLIM : system library management system

S-PL/H & MASS : super programming language and macro assembler for Hitachi microcomputer system
SEMP : source code editing and maintenance program
ORION : online retriever of information

(株)日立製作所のプログラム製品でSEMPはLIMEという製品名で販売している

図 4.28　ソフトウェア開発支援システムの構成　(マイクロコンピュータを利用した大規模ソフトウェア開発に使用されているツール群の全体を示したもので，5個の支援機能から構成されている)



LC：加入者回路
TRK：トランク
SW：スイッチ
ATT：交換台

図 4.29 仮想機械方式の一例

4.5 そ の 他

68000による制御はこれからよりいっそう拡大するものと思われる．そのうち，まだ実用化されていないが見込みの高いものとしてゲーム・マシンと工場全体の制御があげられよう．

16ビット・マイクロプロセッサを使ったゲーム・マシンとして1982年10月に「ぴゅう太」が(株)TOMYから発売された．これは68000によるものではないが，ゲームの内容が複雑となれば，現在の6502や6800などの8ビット・マイクロプロセッサで簡単にカバーすることが難しくなるだろうから，16ビット・マイクロプロセッサへの進出は時間の問題であった．68000を使ったゲーム・マシンがいずれ登場するかもしれない．

一方，68000は16Mバイト，68020は4Gバイトのメモリをもつことができるので，非常に大規模な制御が可能である．特にメモリ管理ユニットMMUをうまく用いれば，工場全体の生産管理などの制御が十分行なえるはずである．しかし残念ながらその実例を知らない．

従来，大型コンピュータで行なっていたものを68000とその周辺LSIでカバーすれば，コスト・パフォーマンスその他の点で大きく進歩できるものと考えられる．これは逆にマイクロプロセッサ技術の進歩を促すこととなり，もっとよいシステムが出現しよう．

現在のところ68000ファミリを中心とするシステムは16ビット・マイクロプロセッサの世界では最高のように思える．LSIの発展は速いから，これをしのぐVLSIが現われるのにそれほど永い月日を要するとは思えない．筆者もそういう夢を抱いているが，こういう研究はコスト，地の利，協力体制，時勢その他が大きく働き，個人レベルでは大変難しい．適当な指導者を中心としたグループ研究を大事にしていかねばなるまい．「マイクロ」な研究グループではなく，ハードウェア，ソフトウェアを全部含めた「マクロ」なグループによる，限りない，速やかな前進を期待して筆をおくこととする．

索　　引

【ワ行】

欧文見出し

著 者 紹 介

岡本　茂（おかもと・しげる）

生年月日　1930年2月4日

最終学歴　1960年東京教育大学理学研究科博士課程修了

現　　在　茨城大学理学部数学科教授・理学博士

佐藤徳訓（さとう・よしのり）

生年月日　1949年9月7日

最終学歴　1972年弘前大学理学部数学科卒

現　　在　日立入間電子㈱

中島　宏（なかじま・ひろし）

生年月日　1945年8月15日

最終学歴　1969年九州大学工学部動力機械工学科卒

現　　在　日立プロセスコンピュータエンジニアリング㈱

大島邦夫（おおしま・くにお）

生年月日　1947年4月25日

最終学歴　1978年米国ヒューストン大学博士課程修了

現　　在　東京理科大学理工学部数学科講師・Ph. D

ザ 68000

ハードウェア・ソフトウェア・アプリケーション

定価 2400 円

1983年8月1日　初版1刷発行

1983年10月15日　初版4刷発行

著　者　岡本　茂・佐藤徳訓　中島　宏・大島邦夫　©1983

発　行　**共立出版株式会社**／南條正男

東京都文京区小日向4－6－19

電話　東京(03)947局2511番(代表)

郵便番号112／振替口座　東京1－57035番

組　版　緑新社

印　刷　壮光舎

製　本　協栄製本



社団法人
自然科学書協会
会員

検印廃止

NDC 007

ISBN 4-320-02203-3

Printed in Japan

計算機・情報関係書より

共立総合コンピュータ辞典 第2版 …… 山下英男監修 A5判・1440頁
コンピュータ英和・和英辞典 …… 日本ユニバック編 B6判・254頁
コンピュータ用語の基礎知識 …… 酒井重恭著 四六判・274頁
コンピューター契約の仕方 …… 栗田昭平訳 A5判・544頁
計算機の歴史 …… 末包良太他訳 A5判・458頁
電子計算機と情報科学 …… 吉田良教他著 A5判・292頁
コンピュータ・サイエンス '77,'78,'79,'80,'81 …… bit編集部編 B5・180～270頁
対訳・解説 コンピュータ英語 …… 日高哲郎他編 A5判・216頁
論理設計とスイッチング理論 …… 室賀三郎他訳 B5判・380頁
グラフ理論 …… 池田貞雄訳 A5判・370頁
グラフとダイグラフの理論 …… 西関隆夫他訳 A5判・450頁
コンピュータによるグラフ理論の応用 …… 五百井清右衛門他訳 A5判・366頁
マイクロコンピュータの基礎知識 …… 駒宮安男監修 B5判・184頁
電子計算機システム 第2版 …… 酒井重恭著 A5判・228頁
インタプリティング計算機 …… 坂村健訳 A5判・334頁
計算機システムの構造 …… 土居範久訳 A5判・142頁
ALGOL60 講義 …… 米田信夫他著 B5判・142頁
コンパイラのうちとそと …… 島内剛一他著 A5判・186頁
やさしいコンパイラの作り方 …… 中西正和他著 A5判・238頁
オペレーティングシステム言語 …… 西村恕彦他訳 B5判・120頁
新しいデータベース技術 …… 鈴木道夫訳編 B5判・280頁
CODASYLのデータベース …… 西村恕彦他監訳 B5判・194頁
FORTRANデータベース …… 西村恕彦監修 B5判・128頁
分散処理とデータ通信 …… 酒井重恭著 A5判・286頁
プログラム書法 第2版 …… 木村泉訳 A5判・252頁
ソフトウェア入門 …… 阿部圭一著 A5判・270頁
ソフトウェア設計 …… 池浦孝雄他訳 A5判・192頁
ソフトウェア・ツール …… 國井利泰編 B5判・312頁
ソフトウェア作法 …… 木村泉訳 A5判・532頁
ソフトウェア工学 …… 國井利泰監修 B5判・288頁
ソフトウェア・プロダクト工学 …… bit編集部編 B5判・320頁
ソフトウェア評価技法 …… 廣瀬健編 B5判・222頁
プログラミング言語 …… 米田信夫他編 B5判・278頁
APL 8週間 …… 竹下亨著 A5判・286頁
プログラミング言語APL …… 竹下亨著 A5判・320頁
プログラム言語Ada基準文法書〈英文〉 …… bit編集部編 B5判・246頁
プログラム言語Ada …… bit編集部編 B5判・232頁
プログラミング言語C …… 石田晴久訳 A5判・250頁
Pascalプログラミング講義 …… 森口繁一他著 A5判・200頁

Pascalプログラミング対話 …… 森口繁一他著 A5判・304頁
PASCAL入門 …… 川合慧著 A5判・248頁
PASCAL 8週間 …… 黒川利明著 A5判・232頁
初心者のためのPASCAL入門 …… 中村和郎訳 A5判・196頁
FORTRANプログラミング演習 …… 小郷寛他著 A5判・142頁
FORTRAN 8週間 …… 吉田敬一著 A5判・296頁
例題形式FORTRANプログラミング …… 中山隆他著 A5判・206頁
FORTRAN77 TSS プログラミング入門 …… 竹沢照他著 A5判・216頁
COBOL 8週間 …… 日高哲郎著 A5判・260頁
COBOL（プログラム編・文法編） …… 西村恕彦他著 B5・168・186頁
入門BASIC …… 斎藤栴郎訳 B5判・150頁
ベーシック入門 …… 森口繁一監訳 A5判・260頁
BASICプログラミング入門 …… 日本DEC(株)編 B5判・282頁
BASIC文法書 …… 日本DEC(株)編 B5判・298頁
マイコンのためのBASICの使い方 …… 前田英明著 A5判・194頁
コンピュータ図形処理 …… 長江貞彦著 A5判・278頁
研究者と技術者のためのミニコン技術入門 …… 桜井捷海編著 A5判・258頁
ミニコン …… bit編集部編 B5判・280頁
マイコンゲームを楽しもう …… 矢矧晴一郎著 B5判・200頁
パソコン・ツール …… 対馬勝英著 B5判・144頁
パーソナル・コンピュータの使い方 …… 石田晴久編 B5判・368頁
マイクロコンピュータとメモリ …… 日本DEC(株)編 A5判・790頁
マイクロコンピュータ インタフェースハンドブック …… 日本DEC(株)編 A5判・782頁
マイコン プログラム入門 …… 上林憲行訳 A5判・300頁
マイコン・ミニコン工学 …… 岩田倫典訳 A5判・448頁
パズルとマイコン …… 玄光男他訳 A5判・184頁
BASICによるシステム・ダイナミックス …… 小玉陽一著 B5判・152頁
16ビット・マイクロプロセッサ …… bit編集部編 B5判・266頁
16ビット マイクロプロセッサ システム デザイン ハンドブック TMS9995 …… 日本テキサス・インスツルメンツ編 B5判・200頁
マイクロプロセッサと論理設計 …… 奥川峻史他訳 A5判・448頁
コンピュータ・エイデッド・テクノロジ ロボット・CAD CAM・FMS …… 奥村昌之編 A5判・238頁
コンピュータ・アーキテクチャの設計 …… 渡辺勝正訳 A5判・370頁
マイコンOS …… 玄光男他訳 A5判・256頁
CAIシステムI・II …… 木村捨雄他訳 A5・310・278頁
システム構造の設計 …… 酒井重恭他著 A5判・204頁
記憶とデバイス …… 石井治著 A5判・254頁
大規模リアルタイム・システム …… 日本ユニバック他編 A5判・276頁
人工知能ハンドブック（第I巻） …… 田中幸吉他監訳 A5判・528頁
人工知能 コンピュータによるゲーム …… 高原康彦他訳 A5判・304頁

借り出したときは

- 本(ほん)は大切(たいせつ)に保管(ほかん)しましょう。
- 必(かなら)ず期日(きじつ)を守(まも)りましょう。
- よごさないようにしましょう。
- 折目(おりめ)をつけないようにしましょう。
- また貸(か)しをやめましょう。

キハラ No. 1482

ザ 68000